Decentralized Data Management for the Semantic Web

Hala Skaf-Molli

▶ **To cite this version:**

Hala Skaf-Molli. Decentralized Data Management for the Semantic Web. Computer Science [cs]. Université de Nantes, 2017. tel-01618806

HAL Id: tel-01618806
https://hal.science/tel-01618806

Submitted on 18 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Hala SKAF-MOLLI

Mémoire présenté en vue de l'obtention du
Habilitation à diriger des recherches de l'Université de Nantes
Label européen
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Soutenue le 6 Octobre 2017

Decentralized Data Management for the Semantic Web

JURY

Présidente :	**M^me^ Pascale KUNTZ**, Professor, University of Nantes
Rapporteurs :	**M. Bernd AMANN**, Professor, University of Pierre et Marie Curie (Paris 6)
	M. Fabien GANDON, Director de recherche, INRIA Sophia-Antipolis Mediterranée
	M. Philippe LAMARRE, Professor, INSA Lyon
Examinateurs :	**M^me^ Maria ESTHER-VIDAL**, Professor, University of Simón BolívarVenezuela and Fraunhofer IAIS, Germany
	M. Abdelkader HAMEURLAIN, Professor, University of Paul Sabatier, Toulouse
	M. François CHAROY, Professor, TELECOM Nancy - Université de Lorraine

Contents

1 Introduction **4**

2 Distributed Semantic Wikis **9**

2.1 SWOOKI: Highly available semantic wikis . . . 10

2.1.1 System Model . . . 11

2.1.2 Data Model . . . 12

2.1.3 Consistency Model . . . 14

2.1.4 Algorithms . . . 18

2.2 DSMW: Decentralized social semantic wikis . . . 20

2.2.1 System Model . . . 20

2.2.2 Data Model . . . 21

2.2.3 Consistency Model . . . 24

2.2.4 Algorithms . . . 25

2.3 Conclusion . . . 27

3 Co-Evolution between Social and Semantic Web **32**

3.1 Problem Statement . . . 35

3.2 A collaborative Recommender System For Wikipedia Conventions . . . 36

3.3 Evaluation . . . 40

3.4 Conclusion . . . 43

4 SPARQL Queries over Linked data and Deep Web **46**

4.1 semLAV Approach . . . 47

4.2 Algorithms . . . 50

4.3 Evaluation . . . 54

4.4 Conclusion 59

5 Read/Write Linked Open Data 61

5.1 SU-Set: a Conflict-Free Replicated Data Type for RDF Graph 63

5.1.1 Algorithms 63

5.1.2 Evaluating and Optimizing SU-Set 64

5.2 Col-Graph: A Synchronization Algorithm for RDF Fragments 68

5.2.1 Fragment Consistency 68

5.2.2 Algorithms 71

5.3 Conclusion 77

6 Federated SPARQL Queries Processing with Replicated Fragments 80

6.1 Problem Statement 81

6.2 Algorithms 83

6.3 Experimental Study 85

6.4 Conclusion 89

7 Research Directions 92

8 Bibliography 95

8.1 Publications by Hala Skaf-Molli (Categorized) 95

8.2 Publications by other authors 108

1

Introduction

The semantic web is an extension of the web where information has precise meaning, and machines are able to understand the information and perform sophisticated tasks for the users [B+01]. The World Wide Web Consortium (W3C) has defined foundation standards for the Semantic Web: (i) the graph based data model defined by the Resource Description Framework (RDF) [MM04] for representing information. (ii) the SPARQL web based protocol and query language for querying RDF data [P+08], and (iii) the Ontology Languages RDFS, and OWL [P+04] for defining shared knowledge.

The semantic web moves the web from purely syntactic pages to semantic ones that describes precisely entities mentioned in web pages using W3C standards as presented in Figure 1.1.

Many semantic data have been published in the last years using W3C standards, e.g, more than *149,945,033,382* RDF triples are available from 2832 datasets [1]. Different kind of data have been published by different data providers: Scientific Publications, e.g. the DBLP Bibliography Database [2], media, e.g., the Jamendo music repository [3], government agencies, e.g., the UK transport

1. stats.lod2.eu, September, 2016
2. http://dblp.l3s.de/d2r/, September, 2016.
3. http://dbtune.org/jamendo/, September, 2016.

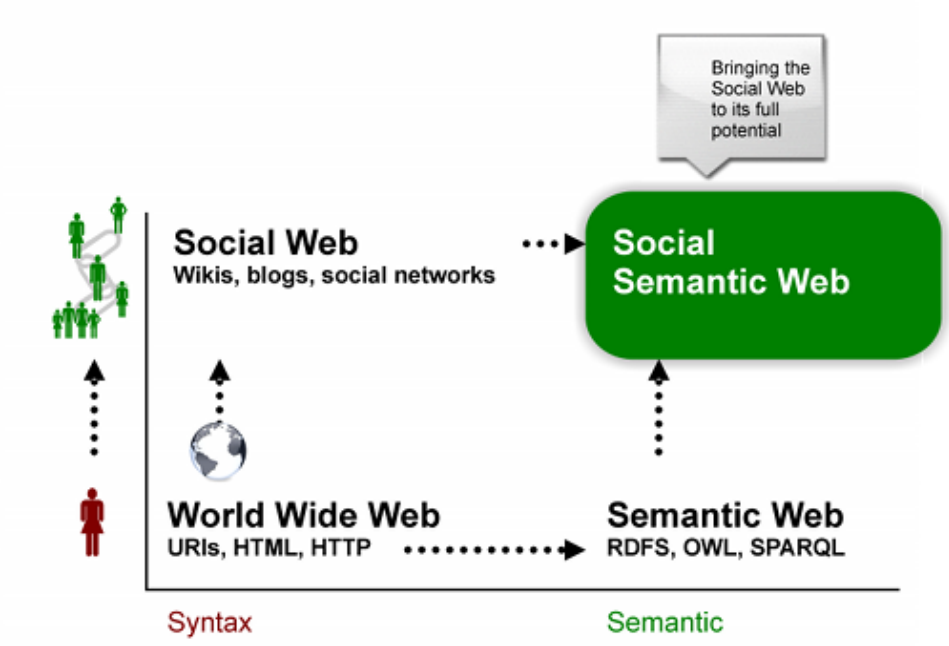


Figure 1.1 – Semantic Web, Social Web and Social Semantic Web: Source [Bre+09]

dataset [4], the Pays de la loire dataset [5], Life Science, e.g., the UniProt dataset [6], Social Networking, e.g., FoaF profiles [7] and the general knowledge dataset, e.g. DBpedia [8].

The published datasets follow the Linked Open Data (LOD) principles [BVS; BHB09]. RDF and ontology are used for representing and linking data from different sources. Linked data forms a decentralized global data graph that could be queried using SPARQL query language [PAG09].

The social web [BPD09] has largely help to bootstrap the semantic web, for instance, DBpedia [Biz+09] knowledge base is built from data extracted from Wikipedia infoboxes and categories. However, the social web does not really benefit from the semantic web. The co-evolution between social and semantic web is an open issue. Among problems to solve are: *Issue1: Which social tools can be used for collaborative production of semantic data? Issue2 : How the social web can take benefits of the semantic web?*.

Despite a great effort has been done by the semantic web community to integrate datasets into the LOD cloud and make these data accessible through SPARQL servers. There are still a large number of data sources and Web APIs that are not part of the LOD cloud. The Deep Web which has around 500 times the size of the Surface Web [He+07] has not been integrated as part of LOD cloud. Performing SPARQL queries without considering the Deep Web can potentially deliver incomplete

4. http://openuplabs.tso.co.uk/sparql/gov-transport, September 2016.
5. http://lodpaddle.univ-nantes.fr/lodpaddle, September 2016.
6. http://sparql.uniprot.org/, September 2016.
7. http://www.w3.org/People/Berners-Lee/card.rdf, September 2016.
8. http://dbpedia.org/sparql, September 2016.

results. *Issue3: How to answer SPARQL query over Linked Data and Deep Web?*.

LOD data has intrinsic issues related to data quality [Aco+13]. If a data consumer finds a mistake or inconsistency, how can she fix it? This raises the issue of the *quality* of Linked Data, *Issue4: How to improve LOD quality?*.

Federated SPARQL query engines [Sch+11; Aco+11] allow data consumers to execute SPARQL queries against a federation of SPARQL servers. In presence of replication, the state-of-art federated query engines [Sch+11; Aco+11] may retrieve data from every relevant server, and produce a large number of intermediate results. Therefore, federated query engines may exhibit poor *performance* while *availability* of the selected SPARQL servers is negatively impacted. *Issue5: How to ensure performance of federated SPARQL query engines with replicated data?*.

In this HDR thesis, I will present my contributions to the above highlighted issues. Chapters 2-6 details these contributions. Each chapter describes motivations, scientifiques contributions, and related thesis, publications and projects. The last chapter presents my ongoing research project. The thesis is organized as follows.

Chapter 2 (Distributed Semantic Wikis). This chapter presents my contributions concerning *Issue1*. I proposed *distributed semantic wiki systems for collaborative editing of RDF data*. These systems rely on optimistic replication approach. Semantic wikis define a new datatype: wiki text that embeds RDF data. The research questions concern consistency criteria for this new datatype. The Causality, Consistency and Intention (CCI) consistency criterion used in collaborative editing systems is not tailored for RDF data model. This chapter details replication models, consistency criteria and algorithms for my two contributions: Swooki and DSMW. Swooki is an unstructured peer-to-peer semantic wiki and DSMW is a decentralized social semantic wiki.

Chapter 3 (Co-Evolution between Social and Semantic Web). This chapter details my contributions related to *Issue2*. Despite DBpedia data are retrieved from Wikipedia, the semantic capacities of DBpedia enable SPARQL queries to retrieve information that are not present in Wikipedia. A SPARQL query that retrieves people born in a place could include more people than those obtained by navigating from the place article in Wikipedia. The integration of missing navigational paths in Wikipedia is very challenging, it requires to respect the complex Wikipedia conventions. I formulate this problem as a recommendation problem, the recommander suggests possible links between wikipedia articles based on similar articles linked by the Wikipedia community. I proposed *a*

collaborative recommender system for enriching the social web with the semantic web. BlueFinder is a collaborative recommender system that enhances the content of Wikipedia with knowledge inferred in DBpedia and contributes to complete the virtuous cycle of information flow between Wikipedia and DBpedia. This chapter presents the information gap between DBpedia and Wikipedia, details the BlueFinder algorithm and an empirical evaluation.

Chapter 4 (Querying Deep web and Linked Data). This chapter details my contributions related to *Issue3*. This chapter details semLAV, the first scalable LAV-based approach for SPARQL query processing. Contrary to traditional LAV approaches, semLAV does not require query rewritings. In semLAV relevant views are ranked according to their possible contribution to the answers, and they are loaded into a graph instance, built during query execution, to answer SPARQL queries. The challenge is to define the order in which query relevant views should be loaded to outperform the traditional LAV query rewriting techniques in terms of number of answers produced by a time unit. This chapter presents semLAV approach, algorithms and experimental evaluation.

Chapter 5 (From Read-Only to Read/Write LOD). For handling *Issue4*, I proposed to transform *the Read-Only LOD into Read/Write LOD*. A data consumer can improve LOD quality by editing the data and push the enhanced data to the data provider. To support a Read/Write LOD, we follow an optimistic replication approach. The scientific problem is to propose new consistency criteria and synchronization algorithms for RDF data and SPARQL updates. This is challenging because of the autonomocity of data providers and data consumers. For instance, a data provider could not accept an update of untrusted data consumer. This chapter details SU-Set and Col-Graph two replication models for Read/Write LOD, their correction criterion and the synchronization algorithm. SU-Set is a Conflict-Free Replicated Type (CRDT) for RDF Graph, it designed for large scale replication of RDF graphs with strong eventual consistency, and Col-Graph is designed for partial (fragment) data replication, it uses annotated RDF graphes and Updates to achieve fragments consistency.

Chapter 6 (Replication-aware Federated Query Engine). This chapter adresses *Issue5*. The research questions are: Can the knowledge about replicated data fragments be used to reduce the number of selected sources by federated query engines while producing the same answers? Does considering groups of triple patterns to be executed together, instead of individual triple patterns, produce source selections that lead to transfer less data from endpoints to the federated query engine?

This chapter presents Fedra, the first source selection strategy for a replicated-aware federated query engine. Fedra uses fragment containment to prune sources that provide redundant data, and an heuristic for set covering to reduce the number of selected data sources.

Chapter 7 (Conclusion and perspectives). Finally, I present my forthcoming work, *Linked Data in the Fog*. The objective is to build a decentralized federated SPARQL query engine for the fog. This engine will run on a network of end-user browsers sharing CPU, caches resources and common interests. The new engine will break the state-of-art tradeoffs between availability and performances thanks to the federation of data consumers.

2

Distributed Semantic Wikis

Motivations : Semantic wikis are a new generation of social 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. Many semantic wikis are proposed [Krö+07; Buf+08]. Despite their success, they suffer from many issues, such as: (i) Scalability and cost: as the number of users and contributions increase, so do the storage and bandwidth requirements. (ii) Centralized control/single point of failure: a centralized wiki is controlled by a single organization. If the organization disappears all the knowledge contributed to the wiki may disappear as well; (iii) No offline access: if a user's internet connection or the central wiki system are temporarily unavailable, the user cannot work; (iv) Handling transactional changes: a user cannot modify multiple pages simultaneously before making modifications visible to other users. All incremental modifications are immediately visible to other users. In some cases,

users may observe inconsistent states of the wiki, as in a database system without transactional support [Rah+09a]; (v) Handling disagreements: even when changes to a page are not made with malicious intent, they can still lead to "edit wars" when different groups of users disagree on the content that is being contributed and they attempt to cancel or override the other groups' input.

Contributions : I defined two distributed semantic wikis models: Swooki and DSMW. Swooki is the first a peer-to-peer semantic wiki and DSMW is the first socially distributed semantic wikis. Swooki mainly tackles issues traditionally related to distributed systems and DSMW is tailored for social topologies.

(1) Swooki *a Highly available semantic wiki.* In Swooki [Rah+09b; RSM09b; SRM08; RSM08a; RSM09b] the infrastructure is decentralized between the participants, but for the users, the system's functionality is identical to that of a traditional wiki. The decentralized infrastructure allows the participants to share the cost of storage, bandwidth, and maintenance and provides availability, scalability, and fault-tolerance for the wiki.

(2) DSMW *a Decentralized social semantic wikis.* DSMW relies on an explicit social network of participants and aims to support the multi-synchronous collaboration model [Dou95] and offline access. In these systems, the collaborative editing process follows cycles of divergence/convergence, where users publish their changes, and acquire those published by others, at the time of their choosing. In addition, users can be selective in the subset of changes that they integrate. However, these changes are integrated automatically by algorithms that enforce particular consistency models, which limit the freedom of the users to select changes from others.

In the following sections, I will detail Swooki, and DSMW.

2.1 Swooki: Highly available semantic wikis

Swooki [SRM09] is the first peer-to-peer semantic wiki. Swooki relies on a self-organized unstructured peer-to-peer network. The collection of wiki pages is *fully replicated* across all the participants' sites. The total replication scheme requires that all peers have the same storage capability. The users are connected to one peer and interact with the local page replicas as with traditional wikis. Users are able to work even when their node is disconnected from the rest of the network. When the network is connected, changes are automatically propagated to the other nodes.

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.

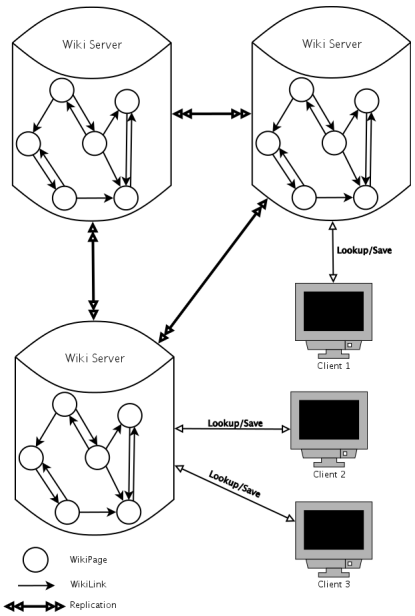


Figure 2.1 – Swooki, Replicated graph in an unstructured P2P wiki

2.1.1 System Model

An unstructured highly available wiki is conceptually similar to a set of n interconnected and automatically synchronized wikis.

Definition 2.1.1. *A* highly available unstructured wiki *is a tuple,* $\langle \Omega_G, N \rangle$ *where:*

- $\Omega_G = \{G_1, G_2, \ldots, G_n\}$ *is a set of wiki graphs;*
- $N = \langle S, C \rangle$ *is a graph representing an unstructured and self-organized overlay network: the nodes* $S = \{S_1, \ldots, S_n\}$ *are participants, each participant is uniquely identified and the edges* $C \subset S \times S$ *represent their physical interconnections. Participant* S_i *hosts* G_i*;*
- *The wiki graphs* G_i *are* eventually consistent*: this notion is defined below.*

The wiki appears centralized because the participant directly interacts only with the local system, which is the wiki $\langle S_i, G_i \rangle$. Propagation of updates happens behind the scenes, and to the user is indistinguishable from operations that might happen concurrently on the local wiki if it was an isolated system. Eventually, the set of local page replicas at each node should converge to be identical. As we will see further, ensuring that this happens is difficult. In order to define eventual consistency, we must consider a highly available unstructured wiki to be a system that evolves over time as a result of the user's actions. We note $G_i^{(t)}$ the state of a graph G_i at time t.

Definition 2.1.2 (Eventual Consistency)**.** *Let* W *be a highly available unstructured wiki,* $W =$

$\langle \Omega_G, N \rangle$ as defined in 2.1.1. We consider a finite sequence of (arbitrary) user actions, occurring at times $t_1, t_2, \ldots t_k$.

The wiki graphs $\{G_i\}_{i \in [1 \ldots n]}$ are eventually consistent *if at some time later than the last action A_{t_k}, all of the graphs G_i are identical. Formally,*

$$\exists t_f > t_k, \forall i, j \in [1 \ldots n] \; G_i^{(t_f)} = G_j^{(t_f)}$$

A highly available unstructured Wiki follows the optimistic replication model [SS05a], with the hypothesis of eventual delivery of operations; this is generally achieved by using the gossiping algorithm. An anti-entropy algorithm supports intermittent connections. Figure 2.1 shows an example of an unstructured wiki. In this figure, the wiki graph is replicated on each wiki server. Each wiki server hosts a copy of all semantic wiki pages and an RDF store for the semantic data. When a peer updates its local copy of data, it generates a corresponding operation. This operation is processed in four steps:

1. It is executed immediately against the local replica of the peer, 2. it is broadcasted through the P2P network to all other peers, 3. it is received by the other peers,

4. it is 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) [Sun+98] consistency model.

2.1.2 Data Model

The data model of SWOOKI is an extension of Wooki [WUM07] 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, I use a string as page identifier.

Definition 2.1.3. *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 2.1.4. *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.

— *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.

— *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]
```

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.

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.

2.1.3 Consistency Model

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

Causality preservation

The causality property ensures that operations ordered by a precedence relation will be executed in the same order on every server.

We define causality for editing operations that manipulate text and RDF data model as:

Definition 2.1.5. ***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 2.1.6. ***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.

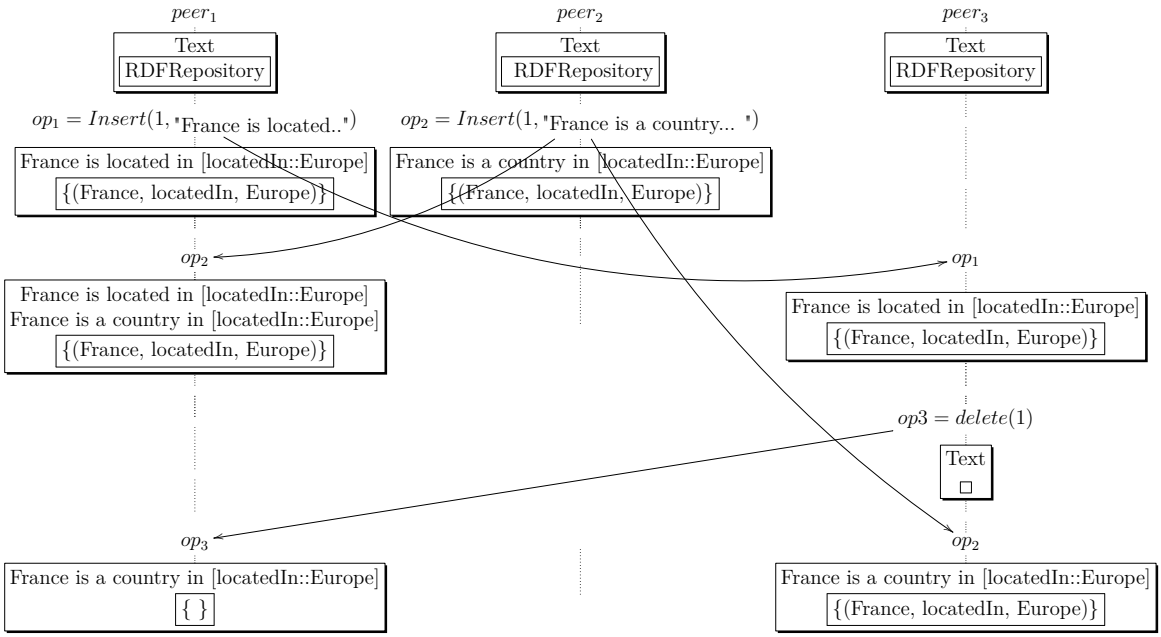


Figure 2.2 – Semantic inconsistency after integrating concurrent modifications

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 2.2. 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.

The above intentions 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.

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 2.1.7. ***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 2.1.8. ***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.*

$$\exists i \quad \wedge \exists i_P < i \; LineID(Page[i_P]) = p \tag{2.1}$$

$$\wedge \exists i \leq i_N \; LineID(Page[i_N]) = n \tag{2.2}$$

$$\wedge Page'[i] = newline \tag{2.3}$$

$$\wedge \forall j < i \; Page'[j] = Page[j] \tag{2.4}$$

$$\wedge \forall j \geq i \; Page'[j] = Page[j-1] \tag{2.5}$$

$$\wedge R' \leftarrow R \uplus T \tag{2.6}$$

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 2.1.9. ***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.*

$$\exists i \qquad \wedge \; PageID(Page'[i]) = ld \tag{2.7}$$

$$\wedge \; visibility(Page'[i]) \leftarrow false \tag{2.8}$$

$$\wedge \; R' \leftarrow R - T \tag{2.9}$$

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.

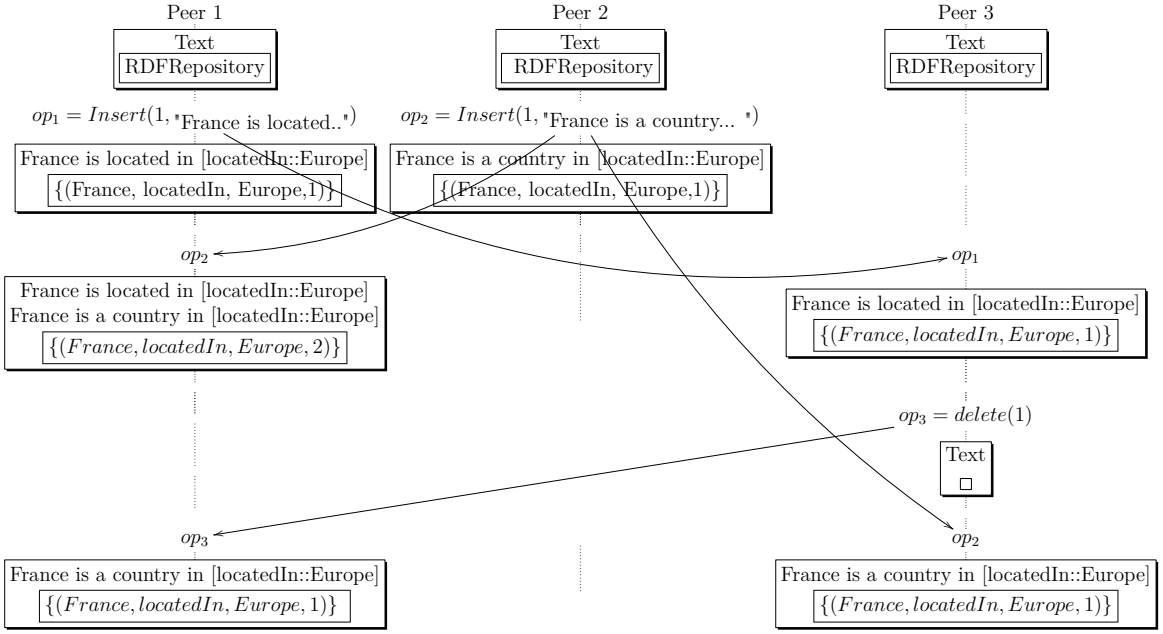


Figure 2.3 – Convergence after integrating concurrent modifications

Let us consider again the scenario of the figure 2.2. 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.

2.1.4 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. In the following, I detail the integration operation.

Algorithm 1 IntegrateDel operation

```
Require: id: LineID;
1: procedure IntegrateDel(id)
2:     IntegrateDelT(id)
3:     IntegrateDelRDF(id)
4: end procedure
```

Algorithm 2 IntegrateIns Operation

```
Require: p: PageID, l,l_P , l_N: LineID;
1: procedure IntegrateIns(p, l, l_P , l_N)
2:     IntegratedInsT(p, l, l_P , l_N)
3:     IntegrateInsRDF(l)
4: end procedure
```

Algorithm 3 IntegrateDelRDF operation

```
Require: id: LineID;
1: procedure IntegrateDelRDF(id)
2:     let S ← ExtractRDF(LineID)
3:     if S ≠ ∅ then
4:         for each triple ∈ S do
5:             triple.counter–
6:             if triple.counter = 0 then
7:                 deleteRDF(R,triple)
8:             end if
9:         end for
10:    end if
11: end procedure
```

Algorithm 4 IntegrateInsRDFOp

```
Require: line: LineID;
1: procedure I(n)tegrateInsRDF(line) :
2:     let S ← ExtractRDF(line)
3:     if S ≠ ∅ then
4:         for each triple ∈ S do
5:             if Contains(triple) then
6:                 triple.counter++
7:             else
8:                 insertRDF(R,triple)
9:             end if
10:        end for
11:    end if
12: end procedure
```

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

For text integration, we use integration algorithm defined in [WUM07]. 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.

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 as described in Algorithm 3

During the insert operation, 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 described in Algorithm 4.

I have integrated this algorithm in Swooki. Swooki is implemented in Java as servlets in a Tomcat Server and uses Sesame 2.0 as RDF store. Swooki is available at: `http://sourceforge.net/projects/wooki/files/`.

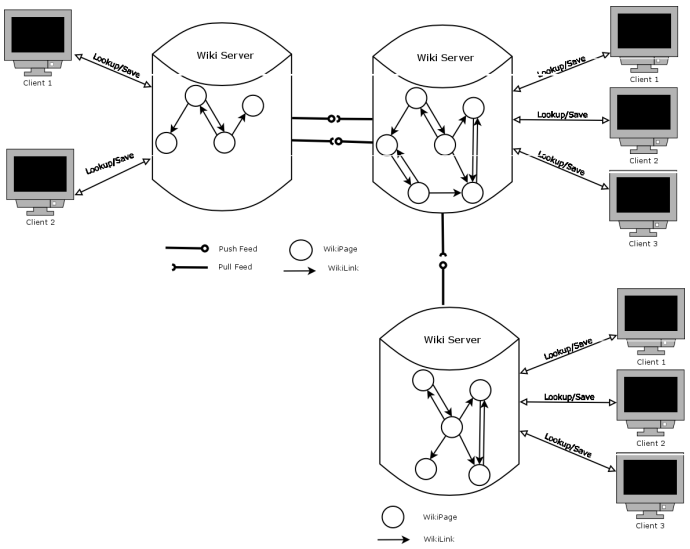


Figure 2.4 – DSMW, a Decentralized Social Wiki

2.2 DSMW: Decentralized social semantic wikis

Decentralized social semantic wikis [Rah+09a] aim to support a social collaboration network and adapt many ideas from decentralized version control systems (DVCS) used for software development. They promote the *multi-synchronous* collaboration model [Dou95], in which multiple streams of activity proceed in parallel. The main structure of a decentralized social wiki is similar to that of a replicated wiki; however, the unstructured overlay network is a social collaboration network: its edges represent relationships between users who have explicitly chosen to collaborate.

The synchronization of the nodes is not fully automated; instead, users can choose pages to replicate and manually publish changes, including sets of changes affecting multiple pages. The changes are propagated along the edges of the social network, and users can select which changes to integrate.

As the published changes are propagated through the network, each wiki graph incorporates a subset of the global sequence of changes, filtered through the participants' trust relationships. The task of integrating selected changes can be automated by algorithms that may enforce different consistency models, as in *highly available wikis*.

The explicit collaboration network and the manual publishing and integration of changes define the class of *decentralized social wikis*, an extension to the main wiki concept.

2.2.1 System Model

Definition 2.2.1. *A* decentralized social wiki *is a tuple* $\langle \Omega_G, N \rangle$*, where:*

— $\Omega_G = \{G_1, G_2, \ldots, G_n\}$ *is a set of wiki graphs, as in a highly available unstructured wiki;*

— $N = \langle S, C \rangle$ *is a graph representing a socially organized overlay network: the nodes* $S = \{S_1, \ldots, S_n\}$ *are uniquely identified participants, and the edges* $C \subset S \times S$ *represent social connections through which operations are exchanged. Each participant* S_i *hosts* G_i*.*

The social connections can be defined as “follow and synchronize" relationships [Rah+09a], in which a user can follow the changes made by specific peers and periodically integrate some or all of these changes. Decentralized social wiki systems may provide a full-blown publish-subscribe protocol (e.g., DSMW, which uses “feeds,” shown in figure 2.4), or simply a social acquaintance relationship that underlies the fact that the “follower” regularly “pulls” (in DVCS terminology) and integrates changes from the “followed” user.

In decentralized social wikis, the content of a wiki graph could be different from one wiki to the next; there is no expectation of consistency at the level of the full wiki. However, automatic synchronization algorithms can still be used; I discuss the consistency issues that they raise in section 2.2.3.

2.2.2 Data Model

The data model of multi-synchronous semantic wikis (M2SW) is defined as an extension of existing ontologies of semantic wikis. Defining the data model as an ontology allows to provides reasoning and querying capabilities on the model itself. In this section, I present the M2SW ontology and detail only its main concepts and their properties.

— **Wiki site** : this concept corresponds to a semantic wiki server. A site has the following properties:

  — *siteID* : this attribute contains the *URL* of the site.

  — *logicalClock* : this attribute has a numeric value. Every semantic wiki server maintains a logical clock, this clock used to identify patches and operations in an unique way in the whole network.

  — *hasPush* : the range of this property is push feed. A wiki site has several push feeds.

  — *hasPull* : the range of this property is pull feed. A wiki site has several pull feeds.

— **Document** : a document can be an image, an audio or a video file or any type of file that can be uploaded in a wiki page.

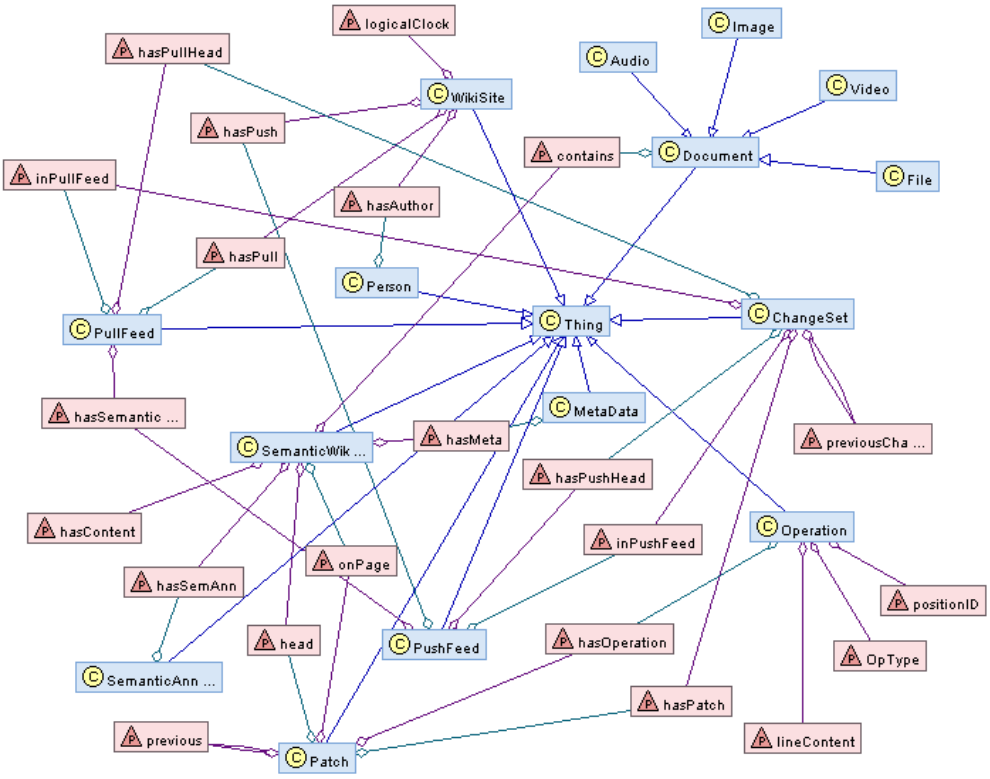


Figure 2.5 – Mutli-synchronous ontology

— **Semantic Wiki page** : this concept corresponds to a normal semantic wiki page. It has the following properties:

  — *pageID* : this attribute contains the *URL* of the page.

  — *hasName* : this attribute contains the *title* of the page.

  — *hasContent* the range of this property is a String, it contains text and the semantic data embedded in the semantic wiki page.

  — *hasPageHead* : this property points to the *last patch* applied to the page.

— **Operation** : represents a change in a line of a wiki page. In our model, there are two editing operations : *insert* and *delete*. An update is considered as a delete of old value followed by an insert of a new value. An operation has the following properties:

  — *operationID*: this attribute contains the unique identifier of the operation. *operationID* is calculated by: $operationID = \text{concat}(Site.siteID, Site.logicalClock++)$. The function *concate* concatenates two strings.

  — *opType* this attribute contains the type of the operation, it can be either an *insert* or a *delete*.

  - *positionID* denotes the position of the line in the wiki page. This identifier is calculated by the *Logoot* algorithm[1].
  - *lineContent* is a string representing text and the semantic data embedded in the line.

- **Patch** : a patch is a set of operations. A patch is calculated during the save of the modified semantic wiki page using the Logoot algorithm. A patch has the following properties:
  - *patchID* a unique identifier of the patch. Its value is calculated by : $patchID = \text{concat}(Site.siteID, Site.logicalClock++)$
  - *onPage* the range of this property is the page where the patch was applied.
  - *hasOperation* this property points to the operations that generated during the save of the page.
  - *previous* points to the precedent patch.
- **ChangeSet** : A change set contains a set of patches. This concept is important in order to support transactional changes. It allows to regroup patches on multiple semantic wiki pages. Therefore, it is possible to push modifications on multiple pages. *ChangeSet* has the following properties:
  - *changSetID*: a unique identifier of a change set. Its value is calculated as : $changeSetID = \text{concat}(Site.siteID, Site.logicalClock++)$
  - *hasPatch* property points to the *Patches* generated since the last push.
  - *previousChangeSet* points to the precedent change set.
  - *inPushFeed* the range of this property is a *PushFeed*. This property indicates the *PushFeed* that publishes a *ChangeSet*.
  - *inPullFeed* the range of this property is a *PullFeed*. This property allows indicates the *PullFeed* that pulls a *ChangeSet*.
- **Push Feed** : this concept is used to publish changes of a *Wikisite*. It is a special semantic wiki page. It inherits the properties of the *Semantic Wiki Page* concept and defines its own properties:

1. Stéphane Weiss, Pascal Urso, and Pascal Molli. « Logoot : a Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks ». In: *32nd International Conference on Distributed Computing Systems*. IEEE Computer Society, 2009, pp. 404–412.

- *hasPushHead* : this property points to the *last* published *changeSet*.
- *hasSemanticQuery* : this property contains a semantic query. This query determines the content of the push feed. For instance, the query can be "find all Lessons", this will return all the page in the class (category) Lessons. To answer *hasSemanticQuery* reasoning and querying capabilities of semantic wikis are used.

— **Pull Feed** : this concept is used to pull changes from a remote $Wikisite$. A pull feed is related to one push feed. In the sense that it is impossible to pull unpublished data. A pull feed is also a special semantic wiki page. It inherits the properties of the *Semantic Wiki Page* concepts and defines it own properties:

— *hasPullHead* : this property points to the last change set pulled in the pull feed.

— *relatedPushFeed*: this property relates a pull feed to the URL of its associated push feed.

We can use this ontology to query and reason on the patches, ChangeSet, PushFeed, ... For instance, we can find published patches on a push feed.

$$Published \equiv \exists(hasPatch^{-1}).\exists(inPushFeed^{-1}).PushFeed$$

The unpublished patches is defined by:

$$unPublished \equiv Patch \sqcap \neg(\exists(hasPatch^{-1}).\exists(inPushFeed^{-1}).PushFeed)$$

2.2.3 Consistency Model

The replication of data and the communication between servers is made through *channels* (feeds). To make local changes available to the others, she *pushes* them to *channels*. The channel usage is restricted to few servers with simple security mechanisms that requires no login and complex access control. The key point is that channels are read-only for consumers and can be hosted on hardware of users. Trusted servers can *pull* these modifications. During the first *pull* operation local copies of pulled pages are created, for the following *pull* operations concurrent modifications are merged with pulled ones.

When a semantic wiki page is updated on a server, it generates a corresponding operation. This operation is processed in four steps as in the case of peer-to-peer semantic wikis.

The system is correct if it ensures the CCI consistency model [Sun+98]. In DSMW, I extend the Logoot [WUM09] synchronization algorithm to be used with the M2SW ontology. Logoot ensures convergence and preserves the intentions of operations if the causality of the operations is preserved.

2.2.4 Algorithms

As any semantic wiki server, a DSMW server defines a *Save* operation which describes what happens when a semantic wiki page is saved. In addition, we have define special operations : *Create Push*, *Push*, *Create Pull*, *Pull* and *Integrate* for the multi-synchronous semantic wiki. We will use the Logoot [WUM09] algorithm for the generation and the integration of the *insert* and *delete* operations.

In the following, I detail these operations for a semantic wiki server called *site*.

Save Operation During the saving a wiki page, the *Logoot* algorithm computes the difference between the saved and the previous version of the page and generates a patch.

Algorithm 5 Save Operation

```
Require: page, page : Wiki Page;
 1: procedure On Save(page, page)
 2:     let Patch p ← Logoot(page, page );
 3:     p.patchID ← concat(site.siteID,site.logicalClock + +);
 4:     for each op ∈ p do
 5:         op.operationID ← concat(site.siteID,site.logicalClock+ +);
 6:         hasOperation(p,op)
 7:         Integrate(op,page)
 8:     end for
 9:     previous(p,page.hasPageHead);
10:     hasPageHead(page,p);
11:     onPage(p,page);
12: end procedure
```

A *patch* is the set of delete and insert operations on the page ($Op = (opType, operationID, positionID, lineContent)$). The *Logoot* algorithm calculates the $positionID$, $lineContent$ and the $opType$ of the operation. These operations are integrated locally and then eventually published on feed patches.

CreatePushFeed Operation The communication between DSMW servers is made through feeds. The *CreatePushFeed* operation allows the creation of a push feed. A push feed is a special semantic wiki page that contains a query that defined the pushed data.

Algorithm 6 Create Push Feed Operation

Require: pfname : FeedName, request : Request

1: **procedure** ON CREATEPUSHFEED(pfname, request)
2: PushFeed(pfname)
3: hasSemanticQuery(pfname,request)
4: P(u)sh(pfname)
5: **end procedure**

Algorithm 7 Create Pull Feed operation

Require: pullfeedName: FeedName, url: URL;

1: **procedure** ON CREATEPULL(pullfeedName, url)
2: PullFeed(pullFeedName)
3: relatedPushFeed(pullfeedName,url)
4: PULL(pullfeedName)
5: **end procedure**

A push feed is used to publish changes of a wiki server. Therefore, authorized sites can access the published data. *CreatePushFeed* operation calls the *Push* operation.

Push Operation The *Push* operation creates a change set corresponding to the pages returned by the semantic query and adds it to the push feed. First, it executes the semantic query, the patches of the pages returned by the query are extracted. These patches are added to the change set if they have not been published on this feed push yet.

Algorithm 8 Push Operation

Require: pfname : FeedName
1: **procedure** ON PUSH(pfname)
2: ChangeSet(chgSet);
3: chgSet.changSetID ← concat(site.siteID,site.logicalClock+ +);
4: inPushFeed(chgSet, pfname);
5: let pg ← execQuery(pfname.hasSemanticQuery) ;
6: let qt ← ∅; let pt ← ∅;
7: **for each** p ∈ pg **do**
8: qt ← qt $\bigcup$ ∃ (onPage^{-1}).pg;
9: pt ← pt $\bigcup$ ∃ onPage.p.∃ (hasPatch^{-1}).∃(inPushFeed^{-1}).pfname;
10: **end for**
11: ∀ patch ∈ pt - qt: hasPatch(chgSet, patch);
12: previousChangeSet(chgSet, pfname.hasPushHead);
13: hasPushHead(pfname,chgSet);
14: **end procedure**

CreatePullFeed Operation As the replication of data and the communication between DSMW servers are made through feeds, pull feeds are created to pull changes from push feeds on remote peers to the local peer (see Algorithm 7). A pull feed is related to a push feed. In the sense that it is impossible to pull unpublished data.

Pull Operation The Algorithm 9 details the pull operation. This operation fetches for published change sets that have not pulled yet. It adds these change sets to the pull feed and integrate them to the concerned pages on the pulled site.

Integration operation The integration of a change set is processed as follows. First all the patches of the change set are extracted. Every operation in the patch is integrated in the corresponding semantic wiki page thanks to the algorithm Logoot detailed in [WUM09] .

Algorithm 9 Pull operation

```
Require: pullfeedName: FeedName
1: procedure On Pull(pullfeedName)
2:     while ((cs ← pullfeedName.PageID.get(pullfeedName.headPullFeed, pullfeedName.relatedPushFeed)<>null) do
3:         inPullFeed(cs,pullfeedName);
4:         Call Integrate(cs);
5:         hasPullHead(pullfeedName,cs);
6:     end while
7: end procedure
```

Correction model

Theorem 2.2.2. *Our algorithms ensure the causality.*

Theorem 2.2.3. *Our algorithms ensure the CCI model (Causality, Convergence, Intention).*

The proofs of theorems are detailed in [Rah+09a]. DSMW is implemented as an extension of Semantic MediaWiki. Demos of DSMW [SCM10b] are presented at the conference ESWC2010 *Extended Semantic Web Conference* and DocEngine Conference [SCM10a].

2.3 Conclusion

In this chapter, I presented my contributions on decentralized semantic wikis: Swooki and DSMW. Swooki mainly tackles issues traditionally related to distributed systems and DSMW is tailored for social topologies. In Swooki and DSMW, semantic data is modified as a side effect of text modification. The advantage is that the text is kept synchronized with semantic data embed in the text, but the drawback is the semantic data cannot be modified directly. Consequently, other authoring tools for semantic web such as Protégé or SPARQL update cannot be used safely on the same semantic data authored through a semantic wiki. For future works, I want to develop new synchronization algorithms dedicated for RDF data and Linked Data.

Supervised Master Thesis associated with this chapter

[Tor09] Diego Torres. « Semantic Wikis using Folksonomies ». MA thesis. licenciature (bac+5) Université de la Plata, Argentine, July 2009.

Supervised PhD Thesis associated with this chapter

[Rah10] Charbel Rahhal. « Wikis sémantiques distribués sur réseaux pair-à-pair ». PhD thesis. Université Henri Poincaré, Nancy1, Nov. 2010.

Publications associated with this chapter

Book chapters

[Cor+14] Amélie Cordier, Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Fadi Badra, Julien Cojan, Emmanuelle Gaillard, Laura Infante-Blanco, Pascal Molli, Amedeo Napoli, and Hala Skaf-Molli. « Taaable: a Case-Based System for personalized Cooking ». In: *Successful Case-based Reasoning Applications-2*. Vol. 494. Studies in Computational Intelligence. Springer, Jan. 2014, pp. 121–162. ISBN: 978-3-642-38735-7. DOI: 10.1007/978-3-642-38736-4_7. URL: http://hal.inria.fr/hal-00912767.

International peer-reviewed journals

[Dav+15] Alan Davoust, Hala Skaf-Molli, Pascal Molli, Babak Esfandiari, and Khaled Aslan. « Distributed Wikis: A Survey ». In: *Concurrency and Computation: Practice and Experience* 27 (2015), pp. 2751–2777. DOI: 10.1002/cpe.

[Naj+09] Hala Naja-Jazzar, Nishadi Desilva, Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. « OntoRest: A RST-based Ontology for Enhancing Documents Content Quality in Collaborative Writing ». In: *INFOCOMP Journal of Computer Science* 8.3 (2009), pp. 1–10.

National peer-reviewed journals

[RSM09b] Charbel Rahhal, Hala Skaf-Molli, and Pascal Molli. « SWooki: Un Wiki Sémantique sur réseau Pair-à-Pair ». In: *Ingénierie des Systèmes d'Information* 14.1 (Feb. 2009), pp. 117–140.

International peer-reviewed conferences

[Le+13] Anh-Hoang Le, Marie Lefevre, Amélie Cordier, and Hala Skaf-Molli. « Collecting interaction traces in distributed semantic wikis ». In: *3rd International Conference on Web Intelligence, Mining and Semantics, WIMS '13, Madrid, Spain, June 12-14, 2013*. 2013, p. 21.

[BBS10] Anne Boyer, Armelle Brun, and Hala Skaf-Molli. « Human Computer Collaboration to Improve Annotations in Semantic Wikis ». In: *6th Conference on Web Information Systems and Technologies (Webist 2010)*. Valencia, Spain, Apr. 2010, p. 8. URL: https://hal.inria.fr/inria-00378416.

[BSB10] Armelle Brun, Hala Skaf-Molli, and Anne Boyer. « Raising up Annotations In Pedagogical Resources by Human-Computer Collaboration ». In: *European Distance and E-learning Network (EDEN 2010)*. Budapest, Hungary, Oct. 2010. URL: https://hal.inria.fr/inria-00597285.

[SCM10a] Hala Skaf-Molli, Gérôme Canals, and Pascal Molli. « DSMW: a distributed infrastructure for the cooperative edition of semantic wiki documents ». In: *ACM Symposium on Document Engineering (DocEng 2010) (Demo)*. Manchester, Royaume-Uni: ACM, 2010, pp. 185–186.

[SCM10c] Hala Skaf-Molli, Gérôme Canals, and Pascal Molli. « DSMW: Distributed Semantic MediaWiki ». In: *7th Extended Semantic Web Conference (ESCW 2010) (Demo)*. Vol. 6089. Lecture Notes in Computer Science. Heraklion, Gréce: Springer, 2010.

[Rah+09a] Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, and Stephane Weiss. « Multi-Synchronous Collaborative Semantic Wikis ». In: *10th International Conference on Web Information Systems Engineering (WISE 2009)*. Vol. 5802. Lecture Notes in Computer Science. Poznan, Pologne: Springer, Oct. 2009, pp. 115–129.

[SRM09] Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. « Peer-to-peer Semantic Wikis ». In: *20th International Conference on Database and Expert Systems Applications - DEXA 2009*. Vol. 5690. Lecture Notes in Computer Science. Linz, Autriche: Springer-Verlag, Aug. 2009, pp. 196–213.

[Tor+09b] Diego Torres, Hala Skaf-Molli, Alicia Diaz, and Pascal Molli. « Supporting Personal Semantic Annotations in P2P Semantic Wikis ». In: *20th International Conference on Database and Expert Systems Applications - DEXA 2009*. Vol. 5690. Lecture Notes in Computer Science. Linz, Autriche: Springer Berlin / Heidelberg, Aug. 2009, pp. 317–331.

International peer-reviewed workshops

[Cha+12] Pierre-Antoine Champin, Amélie Cordier, Elise Lavoué, Marie Lefevre, and Hala Skaf-Molli. « User assistance for collaborative knowledge construction ». In: *Workshop on Semantic Web Collaborative Spaces (SWCS), in conjunction with the World Wide Web 2012 International Conference*. Lyon, France: ACM, Apr. 2012, pp. 1065–1074. URL: https://hal.archives-ouvertes.fr/hal-00692091.

[Bla+10a] Alexandre Blansche, Julien Cojan, Valmi Dufour-Lussier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli, and Yannick Toussaint. « TAAABLE 3: Adaptation of ingredient quantities and of textual preparations ». In: *18h International Conference on Case-Based Reasoning - ICCBR 2010, Computer Cooking Contest Workshop Proceedings*. Alessandria, Italie, 2010. URL: http://hal.inria.fr/inria-00526663.

[Bla+10b] Alexandre Blansche, Hala Skaf-Molli, Pascal Molli, and Amedeo Napoli. « Human-machine Collaboration for Enriching Semantic Wikis using Formal Concept Analysis ». In: *5th Workshop on Semantic Wikis Linking Data and People - SemWiki2010*. 2010.

[Bad+09] Fadi Badra, Julien Cojan, Amélie Cordier, Jean Lieber, Thomas Meilender, Alain Mille, Pascal Molli, Emmanuel Nauer, Amedeo Napoli, Hala Skaf-Molli, and Yannick Toussaint. « Knowledge acquisition and discovery for the textual case-based cooking system WIKITAAABLE ». In: *8th International Conference on Case-Based Reasoning - ICCBR 2009, Workshop Proceedings*. Seattle, USA, July 2009, pp. 249–258.

[Cor+09a] Amélie Cordier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli, and Yannick Toussaint. « WIKITAAABLE: A semantic wiki as a blackboard for a textual case-based reasoning system ». In: *SemWiki 2009 - 4rd Semantic Wiki Workshop at the 6th European Semantic Web Conference - ESWC 2009*. Heraklion, Grèce, May 2009.

[Rah+09b] Charbel Rahhal, Stéphane Weiss, Hala Skaf-Molli, Pascal Urso, and Pascal Molli. « Undo in Peer-to-peer Semantic Wikis ». In: *SemWiki' 2009 - 4rd Semantic Wiki Workshop at the 6th European Semantic Web Conference - ESWC 2009*. Heraklion, Grèce, June 2009.

[Tor+09a] Diego Torres, Alicia Diaz, Hala Skaf-Molli, and Pascal Molli. « Personal Navigation in Semantic Wikis ». In: *International Workshop on Adaptation and Personalization for Web 2.0 - AP-WEB 2.0 2009*. Vol. 485. CEUR Workshop Proceedings. Trento, Italie: CEUR-WS.org, June 2009, pp. 148–151.

[RSM08a] Charbel Rahhal, Hala Skaf-Molli, and Pascal Molli. « SWOOKI: A Peer-to-peer Semantic Wiki ». In: *3rd Semantic Wiki Workshop (SemWiki'2008) at the 5th European Semantic Web Conference (ESWC 2008)*. Ed. by Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel. Vol. 360. CEUR Workshop Proceedings. Tenerife, Espagne: CEUR-WS.org, June 2008, pp. 124–126.

[RSM08b] Charbel Rahhal, Hala Skaf-Molli, and Pascal Molli. « SWOOKI: A Peer-to-peer Semantic Wiki ». In: *The 3rd Semantic Wikis workshop, co-located with the 5th Annual European Semantic Web Conference (ESWC), Tenerife, Spain*. 2008.

National peer-reviewed workshops

[Cor+09b] Amélie Cordier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli, and Yannick Toussaint. « WikiTaaable, un wiki sémantique utilisé comme un tableau noir dans un système de raisonnement à partir de cas textuel ». In: *17ème atelier de Raisonnement à Partir de Cas - RàPC 2009*. Ed. by Béatrice Fuchs and Amedeo Napoli. Paris, France, June 2009.

3

Co-Evolution between Social and Semantic Web

Motivations : Semantic web technologies facilitate search and navigation on the web, while they can be additionally used to extract data from the social web, e.g., DBpedia is built with data extracted from Wikipedia infoboxes and categories. Wikipedia links are translated into properties in DBpedia, and they are semantically described using RDF vocabularies, *i.e.*, DBpedia encodes semantics that is not represented in Wikipedia and provides a more expressive representation of Wikipedia links. Therefore, DBpedia allows for retrieving information that is not available in Wikipedia [Tor+12b]. To illustrate, Listing 3.1a presents a SPARQL query named $Q1$ to retrieve people and their born place using `db-prop:birthplace`. A place could be a country, province, city, or state. Nevertheless, if

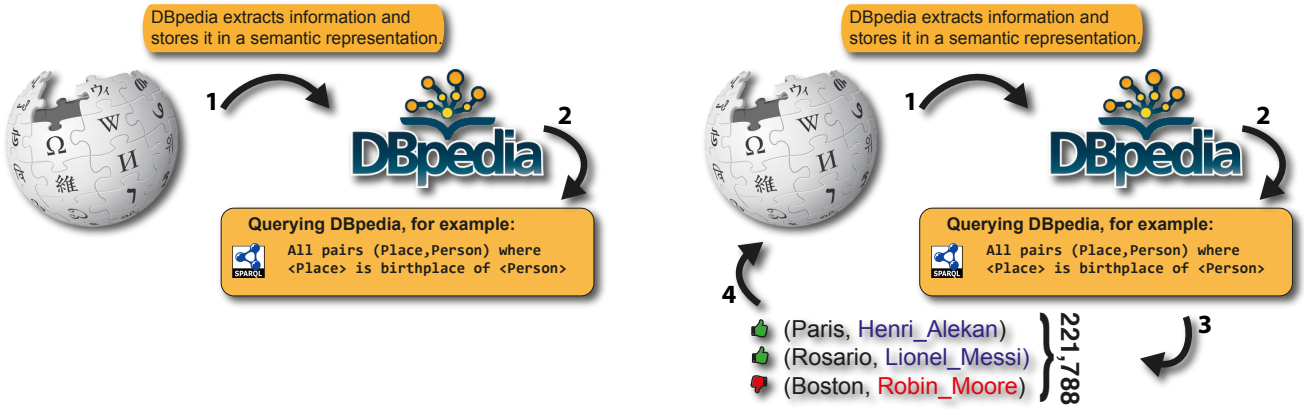
```
select ?city  ?person where {
  ?person a db-o:Person.
  ?city a db-o:City.
  ?person birthplace ?city
}
```

(a) Q1: `is birth palce of`

```
select ?p (count(distinct ?o) as ?count where
    {
        ?s ?p ?o .
        ?s rdv:type dbo:Person  .
        ?o rdv:type dbo:Place .
}
group by ?p
order by ?count
```

(b) Q2: Relevant properties of the classes `Person` and `Place`

Figure 3.1 – DBpedia Queries



(a) Information flow from Wikipedia and DBpedia

(b) Complete cycle of information flow between Wikipedia and DBpedia

Figure 3.2 – Information flow between social web and semantic web

$Q1$ is executed against the DBpedia endpoint [1], the answer includes more people than those obtained by navigating from the Wikipedia place article. The evaluation of query $Q1$ retrieves 409,812 (place, person) pairs from the DBpedia endpoint. Meanwhile, if we navigate from places to people in Wikipedia, we only obtain 221,788 connected pairs. Two Wikipedia articles are connected if a regular Wikipedia user can navigate from one article to another through a navigational path. A navigational path with a length larger than five is unreachable by a regular user [LN85; LC98; OJ00]; so those articles are considered as disconnected. Thus, only 54 % of places in Wikipedia have a navigational path to those people who were born there.

Contributions : In this work, I propose to add missing navigational paths in Wikipedia to enhance Wikipedia content. This is will complete the virtuous cycle of information flow between Wikipedia and DBpedia as illustrated in Figure 3.2b. To measure how important the gap between Wikipedia and DBpedia, we choose the most popular classes of DBpedia defined in [Leh+15], and the properties with the highest number of triples. We call these properties *relevant properties*. Listing 3.1b shows the SPARQL query Q2 that retrieves the relevant properties that relate instances of the classes `dbo:Person` and `dbo:Place`. We observe the same phenomenon when querying DBpedia using *relevant properties* of other classes, e.g., `dbo:Person`, `dbo:Place`, or `dbo:Work`, as shown in Table 3.1. The last two columns of the table provide the number of connected pairs obtained by a SPARQL query and the amount of disconnected pairs in Wikipedia for a specific property, respectively.

1. DBpedia of July 2013

DBpedia Property	from Class	to Class	# DBpedia connected pairs	# Wikipedia disconnected pairs
$prop_1$: birthPlace	Place	Person	409,812	221,788
$prop_2$: deathPlace	Place	Person	108,148	69,737
$prop_3$: party	PoliticalParty	Person	31,371	15,636
$prop_4$: firstAppearance	Work	Person	1,701	142
$prop_5$: recordLabel	Company	Person	25,350	14,661
$prop_6$: associatedBand	MusicalWork	Person	365	73
$prop_7$: Company	Software	developer	14,788	2,329
$prop_8$: recordedIn	PopulatedPlace	MusicalWork	28,351	27,896
$prop_9$: debutstadium	Building	Athlete	595	393
$prop_{10}$: producer	Artist	MusicalWork	70,272	32,107
$prop_{11}$: training	Building	Artist	171	109
$prop_{12}$: previousWork	Album	MusicalWork	72,498	3,887
$prop_{13}$: recordLabel	Company	MusicalWork	118,028	75,329
$prop_{14}$: starring	Person	Film	164,073	42,584
$prop_{15}$: country	PopulatedPlace	Book	19,224	17,281
$prop_{16}$: city	PopulatedPlace	Educational Institution	34,061	8,681
$prop_{17}$: associatedBand	Band	MusicalArtist	24,846	4,100
$prop_{18}$: fromAlbum	Album	Single	18,439	1,268
$prop_{19}$: location	PopulatedPlace	Airport	10,049	2,660
$prop_{20}$: notableWork	Book	Person	1,510	73

Table 3.1 – Results of 20 SPARQL queries for 20 properties and different classes.

Some connected resources in DBpedia are disconnected in their corresponding Wikipedia articles, i.e., resources can be navigated in DBpedia while it is not possible to navigate equivalent resources in Wikipedia. We call this missing navigational paths *information gap* between Wikipedia and DBpedia. Figure 3.3 details the number of information gap between Wikipedia and DBpedia for the properties detailed in Table 3.1.

In order to evaluate the usefulness of adding these navigational paths, we carry out a social evaluation [Tor+12a]. In this evaluation, we have manually added missing navigational paths for 211 disconnected pairs and after one month, we analyzed the number of navigational pairs accepted or rejected by the Wikipedia community. As detailed in [Tor+12a], 90% of new navigational paths were accepted and 10% were rejected. Although the rejected navigational paths had respected the semantics of the relation, they were more general than those used by the community. For example, the proposed navigational path to connect *(Edinburgh, Charlie Aitken)*[2] with the DBpedia property *is birthplace of* was `Edinburgh / Category:Edinburgh / Category:People_from_Edinburgh / Charlie_Ait`[3] Wikipedia community argued that the category *People from Edinburgh* is too general and the more specific category *Sportspeople from Edinburgh* is a more appropriate link.

For adding missing navigational paths, it is mandatory to study how the Wikipedia articles are connected respecting the Wikipedia conventions[4]. *How to find the Wikipedia convention for a*

2. Charlie Aitken (footballer born 1942)
3. It must be read as *from Edinburgh article, the user navigates through a link to the category Edinburgh then he or she navigates to People from Edinburgh category, and then to Charlie Aitken article".*
4. http://en.wikipedia.org/wiki/Wikipedia:Conventions

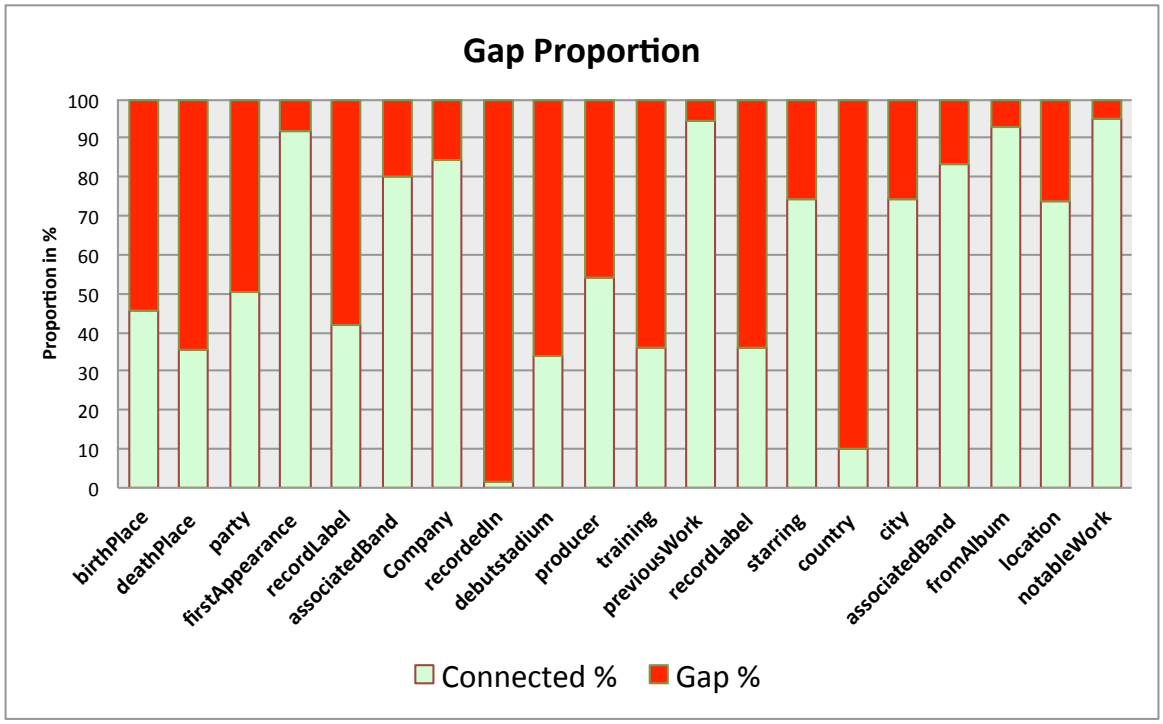


Figure 3.3 – Gap proportion for the twenty DBpedia properties of Table 3.1

navigational path? We formulate the problem of predicting the best representation of missing links in Wikipedia as a collaborative recommender system problem and define BlueFinder, a collaborative recommender system as a solution to this problem. In the following, I detail problem statement and BlueFinder approach and algorithms.

3.1 Problem Statement

According to Adomavicioius and Tuzhilin [AT05], *"collaborative recommender systems try to predict the utility of items for a particular user based on the items previously rated by other users"*. BlueFinder predicts links between Wikipedia articles based on links previously rated by the Wikipedia community. Thus, BlueFinder can be considered as a collaborative recommender system for enhancing content of Wikipedia.

More formally, the utility function $u(c, s)$ of item s for user c is estimated based on the utilities $u(c_j, s)$ assigned to item s by those users $c_j \in C$ who are "similar" to user c. In the context of Wikipedia, BlueFinder does not directly apply recommenders to suggest Wikipedia articles to users but to suggest links between articles. BlueFinder predicts the utility of path queries for a particular pair of Wikipedia articles based on those rated by the Wikipedia community. In other words, the pairs of articles *(from,to)* will play the role of users and the path queries will be the items. Then, the

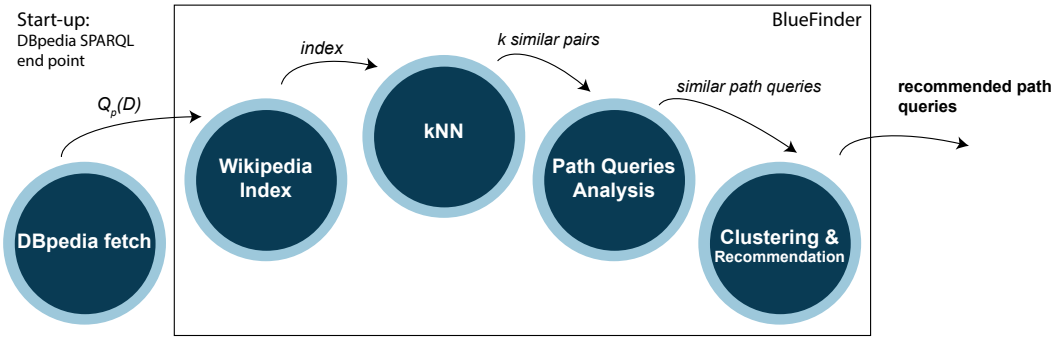


Figure 3.4 – BlueFinder algorithm steps

utility $u(c, pq)$ of a path query pq for a pair c related by a semantic property p is estimated based on the utilities $u(c_j, pq)$ assigned to pair c by those pairs $c_j \in C_p(l)$, $u : Q_p(D) \times PQ \rightarrow R$, where R is a list of path queries sorted according to the rating (see Definition 8).

Given a property p in DBpedia, $C_p(l)$ and PQ path queries covered by the elements of $C_p(l)$. Then, for a given pair of Wikipedia articles $(from, to)$, we have to recommend the path query that maximizes the utility function. The following use case illustrates this problem statement in a practical use.

3.2 A collaborative Recommender System For Wikipedia Conventions

BlueFinder implements a four-steps pipeline process as shown in Figure 3.4. A preprocessing step *DBpedia fetch* configures the BlueFinder start-up information. It fetches from DBpedia SPARQL endpoint the set of pairs of Wikipedia articles $Q_p(D)$ that are related in DBpedia by a semantic property p. After having the $Q_p(D)$, BlueFinder algorithm is ready to start.

The BlueFinder Algorithm 10 receives five inputs: (1) the unconnected pair of Wikipedia articles x, (2) maximum number of recommendations $maxR$, (3) the $Q_p(D)$ set generated by *DBpedia fetch* step, (4) the number of neighbors k , and (5) the maximum length of a path l. BlueFinder algorithm starts by invoking the *WikipediaIndex*.

The *WikipediaIndex* Algorithm 11 builds a path index. It receives $Q_p(D)$ and computes the

Algorithm 10 BlueFinder

Require: x : unconnected pair, $maxR$: maximum number of recommendations, $Q_p(D)$, k : number of neighbors, l:max path length
Ensure: Recommendation path query set
1: $index = (PQ, C_p(l), I) \leftarrow WikipediaIndex(Q_p(D), l)$
2: $k_{neighbors} \leftarrow kNN(x, C_p(l))$
3: $knnPQ \leftarrow \bigcup_{c_i} pq : (pq, c_i) \in I, c_i \in k_{neighbors}$
4: $knnI \leftarrow \bigcup_{c_i} (pq, c_i) : (pq, c_i) \in I, c_i \in k_{neighbors}$
5: $knnPI \leftarrow (knnPQ, k_{neighbors}, knnI)$
6: $M \leftarrow NoiseFilter(knnPI)$ ▷ M ordered in rating descendent order
7: $M \leftarrow StarGeneralization(M, knnPI)$
8: return $maxR$ path queries of M

item set, user set and item ratings. The items are the path queries, and the users are the pairs of Wikipedia pages retrieved from DBpedia. In this case, for each pair of Wikipedia articles $(from, to)$ included in a given $Q_p(D)$, the algorithm performs a depth-first search up to l starting from the $from$ article and finishing in the to article in the Wikipedia graph (lines 1-4). For each reaching to article, it generalizes a path and builds the path index as a bipartite graph (lines 5-8). Finally, it returns the path index that is ready to be used in the next step of the BlueFinder algorithm 10. BlueFinder traverses Wikipedia graph starting from the $from$ article and finishing in the to article in the Wikipedia graph until the maximum length of a path; consequently, a depth-first search is more appropriate than the breadth-first search for building path index.

Algorithm 11 WikipediaIndex

Require: $Q_p(D)$, l: path length
Ensure: PI bipartite graph$Q_p(D)$, l
1: **function** WikipediaIndex($Q_p(D)$, l)
2: $index = (\varnothing, \varnothing, \varnothing)$
3: **for eache**$(from, to) \in Q_p(D)$ **do**
4: $allPaths \leftarrow \varnothing$, $curL \leftarrow 0$, $curPath \leftarrow \varnothing$
5: $GenerateAllPaths(from, to, l, curL, allPaths, curPath)$
6: **for each** $path \in allPaths$ **do**
7: $pathQuery \leftarrow BuildPathQuery(path, from, to)$
8: $index \leftarrow InsertInIndex(index, pathQuery, (from, to))$
9: **end for**
10: **end for**
11: return $index$
12: **end function**

After indexing, BlueFinder performs the kNN step (line 2 in Algorithm 10). In this step, given a disconnected pair of articles in Wikipedia, BlueFinder identifies the k nearest connected pairs to the disconnected one.

The kNN algorithm uses a similarity measure function to select the nearest neighbors. We define

the *Semantic Pair Similarity Distance (SPSD)* function to measure the similarity between pairs of article. SPSD is based on Jaccard distance, it is defined as:

Definition 3.2.1 (Semantic Pair Similarity Distance (SPSD)). *Given two pairs of pages* $c_1 = (a_1, b_1)$ *and* $c_2 = (a_2, b_2)$. *Let* t_{a_1}, t_{b_1}, t_{a_2}, t_{b_2} *data types in DBpedia for* a_1, b_1, a_2 *and* b_2 *respectively. Data types are defined as:*

$t_{a_1} = \{t : < a_1$ *rdf:type* $t > \in$ *DBpedia*$\}$, $t_{b_1} = \{t : < b_1$ *rdf:type* $t > \in$ *DBpedia*$\}$,

$t_{a_2} = \{t : < a_2$ *rdf:type* $t > \in$ *DBpedia*$\}$, $t_{b_2} = \{t : < b_2$ *rdf:type* $t > \in$ *DBpedia*$\}$.

$SPSD(c_1, c_2) = \frac{J(t_{a_1}, t_{a_2}) + J(t_{b_1}, t_{b_2})}{2}$

where J is Jaccard distance between c_1 and c_2. $J(c_1, c_2) = \frac{|c_1 \cup c_2| - |c_1 \cap c_2|}{|c_1 \cup c_2|}$.

To illustrate, we consider two pairs of Wikipedia pages $c1 = (Paris, PierreCurie)$ and $c2 = (Paris, Larusso)$. The data types are:

— $t_{paris} = \{EuropeanCaptialsOfCulture, PopulatedPlace\}$.

— $t_{PierreCurie} = \{Scientist, FrenchCheimists, PeopleFromParis\}$.

— $t_{Larusso} = \{Artist, PeopleFromParis\}$.

$SPSD(c_1, c_2) = \frac{J(t_{paris}, t_{paris}) + J(t_{PierreCurie}, t_{Larusso})}{2} = (0 + 0.75)/2 = 0.375$

Now, we can define the kNN [Lu+12] in our context as:

Definition 3.2.2 (kNN). *Given a pair* $r \in Q_p(D)$ *and an integer* k, *the* k *nearest neighbors of* r *denoted* $KNN(r, Q_p(D))$ *is a set of* k *pairs from* $Q_p(D)$ *where* $\forall o \in KNN(r, Q_p(D))$ *and* $\forall s \in Q_p(D) - KNN(r, Q_p(D))$ *then* $SPSD(o, r) \leq SPSD(s, r)$.

Having the kNN step computed, the *Path Queries Analysis* step starts. It obtains the path queries that connect the k neighbors in a smaller path index than the original (from line 3 to 5 in Algorithm 10).

The generated path index contains the path queries that will be recommended and its ratings. Before the recommendations are returned, in the step *Clustering and Recommendation* in Figure 3.4, BlueFinder cleans regular-user-unreachable-paths (e.g., paths that include administrative categories) by means of the noiseFilter (Algorithm 12) and similar path queries are grouped by StarGeneralization algorithm (Algorithm 13). Finally, BlueFinder returns the $maxR$ best ranked path queries.

The *NoiseFilter* Algorithm 12 deletes all the paths queries that are not accessible by a Wikipedia user. Wikipedia includes several administrative categories which are used by administrators. In order to recommend path queries that can be utilized by regular users, *NoiseFilter* deletes those categories whose names begin with "Articles_", "All_Wikipedia_", etc, such as `Cat:Articles_to_be_merged`.

Algorithm 12 NoiseFilter

Require: $PI = (PQ, C, I)$: Path index
Ensure: Set of regular user navigable path queries.
1: **function** NoiseFilter(PI)
2: $noise = \{"Articles\_", "All\_Wikipedia\_", "Wikipedia\_", "Non-free", "All\_pages\_", "All\_non"\}$
3: **for all** $pq = (p_1, .., p_n) \in PQ$; **do**
4: **if** p_i contains any c $\in noise; 1 \leq i \leq n$ **then**
5: $PQ \leftarrow PQ - \{pq\}$
6: **end if**
7: **end for**
8: return PQ
9: **end function**

BlueFinder filters path queries into star path queries in order to reduce data sparsity.

Algorithm 13 StarGeneralization

Require: PQ: set of path queries, PI: Path index
Ensure: PQ^*: set of star path queries
1: **function** StarGeneralization(PQ, PQ^*)
2: $PQ^* \leftarrow \emptyset$
3: **for each** $pq = (p_1, .., p_{n-1}, p_n) \in PQ$; **do**
4: **if** p_{n-1} starts with "`Cat:`" **then**
5: $PQ^* \leftarrow PQ^* \bigcup \{(p_1, *, p_{n-1}, p_n)\}$
6: **else**
7: $PQ^* \leftarrow PQ^* \bigcup \{pq\}$
8: **end if**
9: **end for**
10: return PQ^*
11: **end function**

Definition 3.2.3. *A star path query $PQ^*(f,t)$ is a group of similar path queries that meet the following rules: (1) $PQ^*(f,t)$ starts with `#from` and ends with `#to`. (2) The * element can only be placed between `#from` and `#to` variables and * represents any path query. (3) The * cannot be the penultimate element in the path query because it has to make explicit the last part of the path in order to make the connection with the `#to` page.*

$PQ^*(f,t)$ =`#from/*/Cat:People_from_#from/ #to` is a star path query.

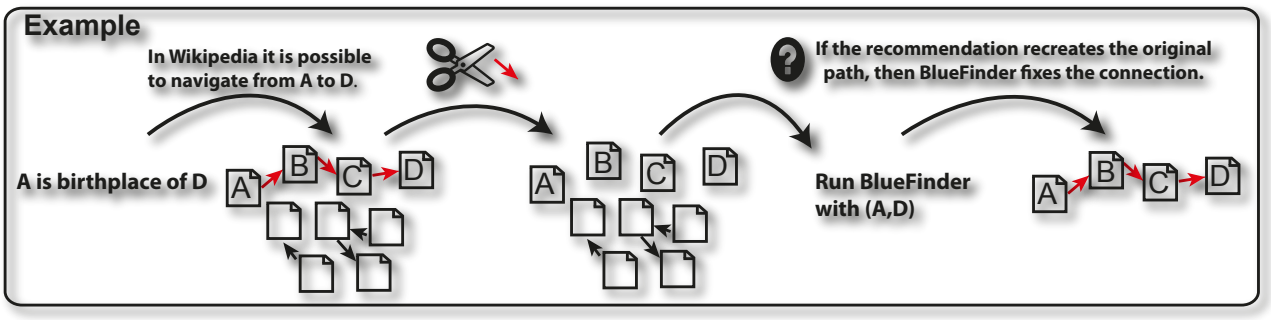


Figure 3.5 – Evaluation method

$PQ^*(f,t) =$`#from/*/#to` is not a star path query.

The *StarGeneralization* Algorithm 13 groups path queries into a star path query, if possible.

3.3 Evaluation

In this section we analyze the behavior of our approach by means of measuring the prediction the accuracy of BlueFinder predictions over the 20 properties shown in Table 3.1. The evaluation is conducted to answer the following questions:

1. What is the best combination of k and $maxR$ values to observe the best accuracy from BlueFinder?
2. Does BlueFinder retrieve path queries that can fix missing relations in Wikipedia?
3. Does the confidence level provided by BlueFinder correlate with the accuracy of the predictions?
4. Does the Wikipedia Community use different conventions to represent a DBpedia property?

In this section we describe the method of the evaluation, the evaluation metrics, and then the data sets used in the experimentation are presented. Finally, the results and discussions are introduced.

Method

In order to answer the questions described in the previous section, an offline evaluation was designed; user interaction is not considered in the study. The central idea of this evaluation is based on disconnecting connected pairs of articles in Wikipedia and then observing whether BlueFinder is able to recreate them. The important fact here is that BlueFinder has to recreate the Wikipedia community conventions that were defined to connect the pairs and not only to discover the disconnection. This approach is based on the assumption that all connected pairs in Wikipedia follow *Wikipedian* conventions. Figure 3.5 summarizes the idea of the evaluation method. For the purpose

of this evaluation all the path queries that connect a pair of pages that are related in DBpedia by a property p, are considered the correct paths that represent the property p.

A sample of 10% of the connected pairs was taken for the evaluation. They are randomly selected and kept in a set called N. For instance, for $prop_1$ in Table 3.1, 188,324 pairs are connected in Wikipedia (i.e. 409,812 - 221,788), so 18,832 randomly selected of those pairs will be in the set N. After that, for each connected pair (w_1, w_2) in N the evaluation repeats the following steps:

1. All paths currently connecting (w_1, w_2) in Wikipedia are stored in the $\mu_{relevant}$ set, and immediately all them are eliminated from Wikipedia to "disconnect" (w_1, w_2).

2. BlueFinder is executed to predict the paths that could connect (w_1, w_2). The resulting predictions are kept in $\mu_{predicted}$.

3. The $\mu_{predicted}$ set is compared with $\mu_{relevant}$ set in order to compute the metrics detailed below such as precision, recall and F1.

4. Finally, Wikipedia is restored up to the state before the pair disconnection. This means that the (w_1, w_2) pair is reconnected by means of $\mu_{relevant}$.

In this evaluation, BlueFinder behavior is evaluated in each property mentioned in Table 3.1, and then aggregates the values of all the metrics to have a general point of view. For example, the evaluation measures the precision metric for $prop_1$, then for $prop_2$ and then it continues with the rest of metrics and properties. After all the metrics and properties are computed, the mean of all metric values is calculated.

In order to have an analysis of the best combination of the number of neighbors and the number of the BlueFinder recommendations, the BlueFinder execution is configured with many combinations of the parameter k and $maxR$ for each disconnected pair. The values for k are from 1 to 10, and the values for $maxR$ are 1, 3, 5 and *unlimited*. The limit of path queries l was fixed in 5 according to the analysis presented previously.

Listing 3.1 – A SPARQL query template for evaluation scenarios

```
select ?fr  ?to where {
        ?db_from a <fromType>  .
         ?db_to a <toType>  .
         ?db_to db-p:<DBpediaSemanticProperty> ?db_from .
         ?db_from foaf:isPrimaryTopicOf ?fr .
         ?db_to  foaf:isPrimaryTopicOf ?to
}
```

Datasets

We evaluate BlueFinder with the twenty semantic properties detailed in Table 3.1. For each property denoted by $prop_i$, a SPARQL query was evaluated on the DBpedia SPARQL endpoint. The SPARQL query for each property follows the template showed in Listing 3.1 and the values of *DBpediaSemanticProperty*, *fromType* and *toType* are replaced in each property scenario for the specific values of the first, second and third column respectively that are detailed in Table 3.1. For instance, the SPARQL query in Listing 3.1a corresponds to $prop_1$. The number of the Wikipedia connected pairs of each property is the difference between the numbers of the DBpedia connected pairs minus the number of the Wikipedia disconnected pairs (columns fourth and fifth of Table 3.1). The evaluation was run with a local copy of the English Wikipedia and DBpedia download in July 2013 and they were stored in a MySQL database.

Evaluation Results

To assess the best behavior of BlueFinder, we analyze the values of accuracy metrics for the 20 properties from a general perspective. Figure 3.6 shows four line-charts with the mean values of *precision*, *recall*, $F1$ and $hit-rate$ obtained for each property. Each chart describes the relation between $maxR$ and k values for each metric. BlueFinder is able to find, on average, between 75 % and 82 % of the relevant paths, and according to the hit-rate values it is able to fix around 88 % of the cases for k greater than 4 and $maxR = 3$, 5 or *unlimited*. However, the limitations is that the precision values decrease according to the variation of the k values and the number of recommendations. To detect the best correlation between precision and recall we use the F1 metric. According to the Figure 3.6, all the $maxR$ curves converge at k=5 with value 0.65. Therefore,

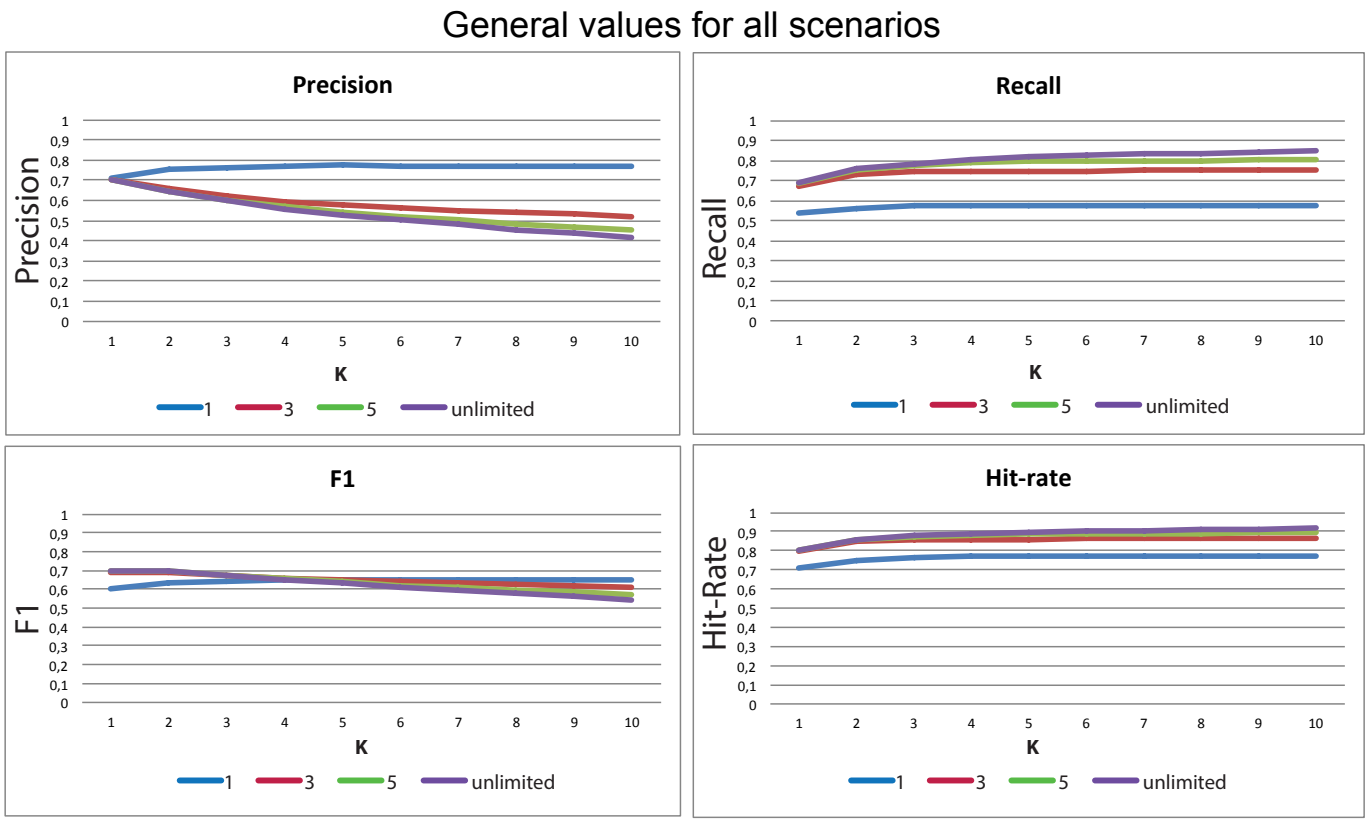


Figure 3.6 – Precision, Recall, F1, and Hit-rate mean of all properties

$maxR = 5$ and $k = 5$ determine the best accuracy for BlueFinder. The number of correct path queries tips the scales in favor of recall and hit-rate rather than precision. This assumption is based on the fact that the recommendations are presented to the users in descending confidence order, and consequently, the users have extra information to determine the accuracy of the recommendation. Finally, the unlimited $maxR$ was dismissed because it had similar recall than $maxR = 5$ but lower precision.

The complete results of all the metrics with the different values for k and $maxR$ are available at the website: https://sites.google.com/site/bfrecommender/publications/.

3.4 Conclusion

In this chapter, I introduced the information gap between Wikipedia and DBpedia. To reduce this gap, we have to discover Wikipedia conventions to represent a DBpedia property between a pair of Wikipedia articles. I proposed BlueFinder, a collaborative recommender system that recommends navigational paths to represent a DBpedia property in Wikipedia, while respecting Wikipedia conventions. BlueFinder learns from those similar pairs already connected by Wikipedia community and proposes a set of recommendations to connect a pair of disconnected articles. BlueFinder exploits

DBpedia types to define a similarity function. Experimental results demonstrate that BlueFinder is able to fix in average 89 % of the disconnected pairs with good accuracy and confidence.

Currently, BlueFinder is tailored for Wikipedia/DBpedia where entities matching are well-defined. However, BlueFinder can be generalized to other datasets with established entities matching.

As a further work, I plan to extend the approach to any property in DBpedia in combination with other languages of Wikipedia and to offre the next generation of BlueFinder as a service for any Wikipedia editor.

Supervised PhD Thesis associated with this chapter

[Tor14] Diego Torres. « Co-Evolution between Social and Semantic Web ». PhD thesis. Université de Nantes, Oct. 2014.

Publications associated with this chapter

Book chapters

[Tor+16] Diego Torres, Hala Skaf-Molli, Pascal Molli, and Alicia Diaz. « Discovering Wikipedia Conventions Using DBpedia Properties ». In: Revised Selected, Invited Papers of Semantic Web Collaborative Spaces: SWCS 2013, and SWCS 2014, Springer International Publishing, 2016, pp. 115–144.

International peer-reviewed conferences

[Tor+13] Diego Torres, Hala Skaf-Molli, Pascal Molli, and Alicia Diaz. « BlueFinder: Recommending Wikipedia Links Using DBpedia Properties ». In: *ACM Web Science Conference 2013 (WebSci 13)*. Paris, France, May 2013, pp. 115–144.

[Tor+12a] Diego Torres, P. Molli, H. Skaf-Molli, and A. Diaz. « From DBpedia to Wikipedia: Filling the Gap by Discovering Wikipedia Conventions ». In: *2012 IEEE/WIC/ACM International Conference on Web Intelligence (WI 12)*. 2012.

[Tor+11] Diego Torres, Alicia Diaz, Hala Skaf-Molli, and Pascal Molli. « Semdrops: A Social Semantic Tagging Approach for Emerging Semantic Data ». In: *2011 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2011)*. Lyon, France, Aug. 2011.

International peer-reviewed workshops

[Tor+12b] Diego Torres, Pascal Molli, Hala Skaf-Molli, and Alicia Diaz. « Improving Wikipedia with DBpedia ». In: *SWCS - Semantic Web Collaborative Spaces Workshop 2012 in 21st WWW Conference 2012*. Lyon, France, Apr. 2012.

4

SPARQL Queries over Linked data and Deep Web

Motivations: A great effort has been done by the Semantic Web community to integrate datasets into the Linking Open Data (LOD) cloud and make these data accessible through SPARQL servers. However, the Deep Web which has around 500 times the size of the Surface Web [He+07] has not been integrated as part of LOD cloud. Performing SPARQL queries without considering the Deep Web can potentially deliver incomplete results. For example, the execution of the SPARQL query: *Which members of the Semantic Web community are interested in* Dalai Lama*,* Barack Obama*, or* Rihanna*?* without the integration of the Deep Web will provide no answers. Nevertheless, if data from social networks such as Twitter, Facebook, or LinkedIn were considered, the query execution could return some answers.

Contributions: I proposed semLAV, the first scalable Local-As-View (LAV) based approach for SPARQL query processing over LOD Cloud and deep web. Given a SPARQL query Q on a set M of LAV views, semLAV selects relevant views for Q and ranks them in order to maximize query results. Next, data collected from selected views are included into a partial instance of the global schema, where Q can be executed whenever new data is included; and thus, semLAV incrementally produces

query answers. In the following, I will present semLAV approach, algorithms and experimental results.

4.1 semLAV Approach

semLAV is a scalable LAV-based approach for processing SPARQL queries. It is able to produce answers even for SPARQL queries against large integration systems with no statistics. semLAV follows the traditional mediator-wrapper architecture [Wie92]. Schemas exposed by the mediators and wrappers are expressed as RDF vocabularies. Given a SPARQL query Q over a global schema G and a set of sound views $M = \{v_1, \ldots, v_m\}$, semLAV executes the original query Q rather than generating and executing rewritings as in traditional LAV approaches. semLAV builds an instance of the global schema on-the-fly with data collected from the relevant views. The relevant views are considered in an order that enables to produce results as soon as the query Q is executed against this instance.

Contrary to traditional wrappers which populate structures that represent the heads of the corresponding views, semLAV wrappers return RDF Graphs composed of the triples that match the triple patterns in the definition of the views. semLAV wrappers could be more expensive in space than the traditional ones. However, they ensure that original queries are executable even for full SPARQL queries and they make query execution dependent on the number of views rather than on the number of rewritings.

To illustrate the semLAV approach, consider a SPARQL query Q with four subgoals given in Listing 4.1, and a set M of five views given in Listing 4.2.

Listing 4.1 – Products, features, and vendor of the offers

```
SELECT * WHERE {
    ?Offer bsbm:vendor ?Vendor .
    ?Vendor rdfs:label ?Label .
    ?Offer bsbm:product ?Product .
    ?Product bsbm:productFeature ?ProductFeature .
}
```

Listing 4.2 – Views that describe contents of five sources having data about products

```
v1(P,L,T,F):-label(P,L),type(P,T),productfeature(P,F)
v2(P,R,L,B,F):-producer(P,R),label(R,L),publisher(P,B),productfeature(P,F)
v3(P,L,O,R,V):-label(P,L),product(O,P),price(O,R),vendor(O,V)
v4(P,O,R,V,L,U,H):-product(O,P),price(O,R),vendor(O,V),label(V,L),offerwebpage(O,U),homepage(V,H)
v5(O,V,L,C):-vendor(O,V),label(V,L),country(V,C)
```

In the traditional LAV approach, 60 rewritings are generated and the execution of all these rewritings will produce all possible answers. However, this is time-consuming and uses a non-negligible amount of memory to store data collected from views present in the rewritings. In case there are not enough resources to execute all these rewritings, as many rewritings as possible would be executed. We apply a similar idea in semLAV, if it is not possible to consider the whole global schema instance to ensure a complete answer, then a partial instance will be built. The partial instance will include data collected from as many relevant views as the available resources allow.

The execution of the query over this partial schema instance will cover the results of executing a number of rewritings. The number of rewritings covered by the execution of Q over the partial schema instance could be exponential in the number of views included in the instance. Therefore, the size of the set of covered rewritings may be even greater than the number of rewritings executable in the same amount of time.

Table 4.1 – Impact of the different views ordering on the number of covered rewritings

# Included views (k)	Order One		Order Two	
	Included views (V_k)	# Covered rewritings	Included views (V_k)	# Covered rewritings
1	v5	0	v4	0
2	v5, v1	0	v4, v2	2
3	v5, v1, v3	6	v4, v2, v3	12
4	v5, v1, v3, v2	8	v4, v2, v3, v1	32
5	v5, v1, v3, v2, v4	60	v4, v2, v3, v1, v5	60

The order in which views are included in the partial global schema instance impacts the number of covered rewritings. Consider two different orders for including the views of the above example: v5, v1, v3, v2, v4 and v4, v2, v3, v1, v5. Table 4.1 considers partial global schema instances of different sizes. For each partial global schema instance, the included views and the number of covered rewritings are presented. Executing Q over the growing instances corresponds to the execution of a quite different number of rewritings. For instance, if only four views could be included with the available

resources, one order corresponds to the execution of 32 rewritings while the another one corresponds to the execution of only eight rewritings. If all relevant views for query Q could be included, then a complete answer will be produced. However, the number of relevant views could be considerably large, therefore, if we only have resources to consider k relevant views, V_k, we should consider the ones that increase the chances of obtaining answers. With no knowledge about data distribution, we can only suppose that each rewriting has nearly the same chances of producing answers. Thus, the chances of obtaining answers are proportional to the number of rewritings covered by the execution of Q over an instance that includes views in V_k.

Maximal Coverage Problem (MaxCov). *Given an integer $k > 0$, a query Q on a global schema G, a set M of sound views over G, and a set R of conjunctive queries whose union is a maximally-contained rewriting of Q in M. The Maximal Coverage Problem is to find a subset V_k of M comprised of k relevant views for Q, $V_k \subseteq M \wedge (\forall v : v \in V_k : v \in RV(Q, M)) \wedge |V_k| = k$, such that the set of rewritings covered by V_k, $Coverage(V_k, R)$, is maximal for all subsets of M of size k, i.e., there is no other set of k views that can cover more rewritings than V_k. $Coverage(V_k, R)$ is defined as:*

$$Coverage(V_k, R) = \{r : r \in R \wedge (\forall p : p \in body(r) : p \in V_k)\} \tag{4.1}$$

The MaxCov problem has as an input a solution to the Maximally-Contained Rewriting problem. Nevertheless, using this for building a MaxCov solution would be unreasonable since it makes the MaxCov solution at least as expensive as the rewriting generation. Instead of generating the rewritings, we define a formula that estimates the number of covered rewritings when Q is executed over a global schema instance that includes a set of views. It is the product of the number of ways each query subgoal can be covered by the set of views. For a query $Q(\bar{X}) \text{ :- } p_1(\bar{X}_1), \ldots p_n(\bar{X}_n)$ using only views in V_k this formula is expressed as:

$$NumberOfCoveredRewritings(Q, V_k) = \Pi_{1 \leq i \leq n} |Use(V_k, p_i(\bar{X}_i))|, \tag{4.2}$$

where $Use(V_k, p) = \Sigma_{v \in V_k} \Sigma_{w \in body(v) \wedge covers(w,p)} 1$. This formula computes the exact number of covered rewritings when all the view variables are distinguished; this is because the coverage of each query subgoal by a given view can be considered in isolation. Otherwise, this expression corresponds to an upper bound of the number of covered rewritings of Q with respect to V_k.

Consider the second proposed ordering of the views in the above example, the numbers of views in V_4 that cover each query subgoal are:

— two for the first query subgoal (v4 and v3),

— four for the second query subgoal (v4, v2, v3 and v1),

— two for the third query subgoal (v4 and v3), and

— two for the fourth query subgoal (v2 and v1).

Thus, the number of covered rewritings is 32 ($2 \times 4 \times 2 \times 2$).

Next, we detail a solution to the MaxCov problem under the assumption that views only contain distinguished variables.

4.2 Algorithms

I will detail two algorithms. The first one selects and ranks relevant views for a query and the seconds builds the global instance to execute the query.

Relevant View Selection and Ranking Algorithm

The relevant view selection and ranking algorithm finds the views that cover each subgoal of a query. This algorithm creates a bucket for each query subgoal q, where a bucket is a set of relevant views; this resembles the first step of the Bucket algorithm[Hal01]. Additionally, the algorithm sorts the buckets views according to the number of covered subgoals. Hence, the views that are more likely to contribute to the answer will be considered first. This algorithm is defined in Algorithm 14.

Algorithm 14 The Relevant View Selection and Ranking

```
Require: Q : SPARQL Query; M: Set of Views defined as conjunctive queries
Ensure: Buckets: Predicate → List<View>
  for all q ∈ body(Q) do
      buckets(q) ← ∅
  end for
  for all q ∈ body(Q) do
      b ← buckets(q)
      for all v ∈ M do
          for all w ∈ body(v) do
              if There are mappings τ, ψ, such that ψ(q) = τ(w) then
                  v_i ← λ(v)          ▷ λ(v) replaces all variables a_i in the head of v by τ(a_i)
                  insert(b, vi)       ▷ add v_i to the bucket if it is not redundant
              end if
          end for
      end for
  end for
  for all q ∈ body(Q) do
      b ← buckets(q)
      sortBucket(buckets,b)           ▷ MergeSort with key (#covered buckets,#views subgoals)
  end for
```

The mapping τ relates view variables to query variables.

The *sortBucket(buckets, b, q)* procedure decreasingly sorts the views of bucket b according to the number of covered subgoals. Views covering the same number of subgoals are sorted decreasingly according to their number of subgoals. Intuitively, this second sort criterion prioritizes the more selective views, reducing the size of the global schema instance. The sorting is implemented as a classical *MergeSort* algorithm with a complexity of $O(|M| \times log(|M|)$.

Proposition 4.2.1. *The complexity of Algorithm 14 is* $Max(O(N \times |M| \times P), O(N \times |M| \times log(|M|)))$ *where* N *is the number of query subgoals,* M *is the set of views and* P *is the maximal number of view subgoals.*

To illustrate Algorithm 14, consider the SPARQL query Q and the previously defined views v1-v5.

Table 4.2 – For query Q, buckets produced by Algorithm 14 when k views have been included. V_k is obtained by Algorithm 15 and the number of covered rewritings

(a) Unsorted buckets

vendor(O,V)	label(V,L)	product(O,P)	productfeature(P,F)
v3(P,L,O,R,V)	v1(P,L,T,F)	v3(P,L,O,R,V)	v1(P,L,T,F)
v4(P,O,R,V,L,U,H)	v2(P,R,L,B,F)	v4(P,O,R,V,L,U,H)	v2(P,R,L,B,F)
v5(O,V,L,C)	v3(P,L,O,R,V)		
	v4(P,O,R,V,L,U,H)		
	v5(O,V,L,C)		

(b) Sorted buckets

vendor(O,V)	label(V,L)	product(O,P)	productfeature(P,F)
v4(P,O,R,V,L,U,H)	v4(P,O,R,V,L,U,H)	v4(P,O,R,V,L,U,H)	v2(P,R,L,B,F)
v3(P,L,O,R,V)	v3(P,L,O,R,V)	v3(P,L,O,R,V)	v1(P,L,T,F)
v5(O,V,L,C)	v2(P,R,L,B,F)		
	v1(P,L,T,F)		
	v5(O,V,L,C)		

(c) Included views

# Included views (k)	Included views (V_k)	# Covered rewritings
1	v4	$1 \times 1 \times 1 \times 0 = 0$
2	v4, v2	$1 \times 2 \times 1 \times 1 = 2$
3	v4, v2, v3	$2 \times 3 \times 2 \times 1 = 12$
4	v4, v2, v3, v1	$2 \times 4 \times 2 \times 2 = 32$
5	v4, v2, v3, v1, v5	$3 \times 5 \times 2 \times 2 = 60$

Algorithm 14 creates a bucket for each subgoal in Q as shown in Table 4.2a.

For instance, the bucket of subgoal $vendor(O, V)$ contains v3, v4 and v5: all the views having a subgoal covering $vendor(O, V)$. The final output after executing the *sortBucket* procedure is described in Table 4.2b.

Views v3 and v4 cover three subgoals, but since v4 definition has more subgoals, i.e., it is more selective, v4 is placed before v3 in all the buckets.

Global Schema Instance Construction and Query Execution

Each bucket is considered as a stack of views, having on the top the view that covers more query subgoals. A global schema instance is constructed as described in Algorithm 15 by iteratively popping one view from each bucket and loading its data into the instance.

Table 4.2c shows how the number of covered rewritings increases as views are included into the global schema instance. Each V_k in this table is a solution to the MaxCov problem, i.e., the number of covered rewritings for each V_k is maximal. There are two possible options regarding query execution. Query can be executed each time a new view is included into the schema instance and partial results will be produced incrementally; or, it can be executed after including the k views. The first option prioritizes the time for obtaining the first answer, while the second one favors the total time to receive all the answers of Q over V_k. The first option produces results as soon as possible; however, in case of non-monotonic queries, i.e., queries where partial results may not be part of the query answer, this query processing approach should not be applied. Among non-monotonic queries, there are queries with modifiers like `SORT BY` or constraints like a `FILTER` that includes the negation of a `bound` expression. The execution of non-monotonic queries requires all the relevant views to be included in the global schema instance in order to produce the correct results.

Algorithm 15 The Global Schema Instance Construction and Query Execution

Require: Q : Query
Require: $Buckets$: Predicate → List<View> ▷ The buckets are produced by Algorithm 14
Require: k : Int
Ensure: A: Set<Answer>
  $Stacks$: Predicate → Stack<View>
  V_k : Set<View>
  G : RDFGraph
  for all $p \in domain(Buckets)$ **do**
    $Stacks(p) \leftarrow toStack(Buckets(p))$
  end for
  while $(\exists p| : \neg empty(Stacks(p))) \wedge |V_k| < k$ **do**
    for all $p \in domain(Stacks) \wedge \neg empty(Stacks(p))$ **do**
      $v \leftarrow pop(Stack(p))$
      if $v \notin V_k$ **then**
        load v into G ▷ only if is not redundant
        $A \leftarrow A \cup exec(Q, G)$ ▷ Option 1: Execute Q after each successful load
        $V_k \leftarrow V_k \cup \{v\}$
      end if
    end for
  end while
  $A \leftarrow exec(Q, G)$ ▷ Option 2: execute before exit

Proposition 4.2.2. *Considering conjunctive queries, the time complexity of Algorithm 15 in option 1 is $O(k \times N \times I)$, while the time complexity is $O(N \times I)$ for option 2. Where k is the number*

of relevant views included in the instance, N the number of query subgoals, and I is the size of the constructed global schema instance.

Proposition 4.2.3. *Algorithm 15 finds a solution to the MaxCov problem.*

Proof. By contradiction, suppose that the set V_k is not maximal in terms of the number of covered rewritings, then there is another set V'_k of size k that covers more rewritings than V_k. By construction, V_k includes the first views of each bucket, i.e., the views that cover more query subgoals. There should exist at least one view in V_k that is not in V'_k, and vice-versa. Suppose w is the first view in V_k that is not in V'_k ($w \in V_k \wedge w \notin V'_k$) , v is the first one in V'_k and is not in V_k ($v \in V'_k \wedge v \notin V_k$) , and w belongs to the bucket of the query subgoal q. If v covers q, then it belongs to the bucket of q. Because V_k includes the views that cover more subgoals, if v was not included in V_k is because it covers less rewritings than w; thus, the contribution of v to the number of covered rewritings is inferior to the contribution of w. This generalizes to all the views in V'_k and not in V_k; thus, the number of rewritings covered by V'_k should be less than the number of rewritings covered by V_k. If v covers another query subgoal q' and all the query subgoals are covered at least once by views in V_k; thus, Algorithm 15 should have included it before including w and v should belong to V_k. □

The semLAV Properties

Given a SPARQL query Q over a global schema G, a set M of views over G, the set RV of views in M relevant for Q, a set R of conjunctive queries whose union is a maximally-contained rewriting of Q using M, and V_k a solution to the MaxCov problem produced by semLAV.

— *Answer Completeness:* If semLAV executes Q over a global schema instance I that includes all the data collected from views in RV, then it produces the complete answer. semLAV outputs the same answers as a traditional rewriting-based query processing approach:

$$\bigcup_{r \in R} r(I(M)) = Q(\bigcup_{v \in RV} I(v)). \tag{4.3}$$

— *Effectiveness:* the *Effectiveness* of semLAV is proportional to the number of covered rewritings, it is defined as:

$$Effectiveness(V_k) = \frac{|Coverage(V_k, R)|}{|R|}. \tag{4.4}$$

For an execution constrained by time or space, V_k could be smaller than RV.

— *Execution Time depends on $|RV|$:* The load and execution time of semLAV linearly depends on the size of the views included in the global schema instance.

— *No memory blocking:* semLAV guarantees to obtain a complete answer when $\bigcup_{v \in RV} I(v)$ fits into memory. If not, it is necessary to divide the set RV of relevant views into several subsets RV_i, such that each subset fits into memory and for any rewriting $r \in R$ all views $v \in body(r)$ are contained in one of these subsets.

4.3 Evaluation

Table 4.3 – Queries and their answer size, number of subgoals, and views size

(a) Query information

Query	Answer Size	# Subgoals	# Rewritings
Q1	6.68E+07	5	2.04E+10
Q2	5.99E+05	12	1.57E+24
Q4	2.87E+02	2	1.62E+04
Q5	5.64E+05	4	7.48E+07
Q6	1.97E+05	3	3.14E+05
Q8	5.64E+05	3	1.57E+05
Q9	2.82E+04	1	3.40E+01
Q10	2.99E+06	3	4.40E+06
Q11	2.99E+06	2	9.25E+03
Q12	5.99E+05	4	1.50E+09
Q13	5.99E+05	2	6.47E+04
Q14	5.64E+05	3	2.52E+06
Q15	2.82E+05	5	2.04E+10
Q16	2.82E+05	3	3.14E+05
Q17	1.97E+05	2	4.62E+03
Q18	5.64E+05	4	1.20E+09

(b) Views size

Views	Size
V1-V34	201,250
V35-V68	153,523
V69-V102	53,370
V103-V136	26,572
V137-V170	5,402
V171-V204	66,047
V205-V238	40,146
V239-V272	113,756
V273-V306	24,891
V307-V340	11,594
V341-V374	5,402
V375-V408	5,402
V409-V442	78,594
V443-V476	99,237
V477-V510	1,087,281

We compare the semLAV approach with traditional rewriting-based approaches GQR [KA11], MCDSAT [ABV06], MiniCon [PH01], and SSDSAT [IVB10]

The Berlin SPARQL Benchmark (BSBM) [BS09] is used to generate a dataset of 10,000,736 triples using a scale factor of 28,211 products. Additionally, third-party queries and views are used to provide an unbiased evaluation of our approach. In our experiments, the goal is to study semLAV as a solution to the MaxCov problem, and we compute the number of rewritings generated by three state-of-the-art query rewriters. From the 18 queries and 10 views defined in [Cas12], we leave out the ones using constants (literals) because the state-of-the-art query rewriters are unable to handle constants either in the query or in the views. In total, we use 16 out of 18 queries and nine out of 10

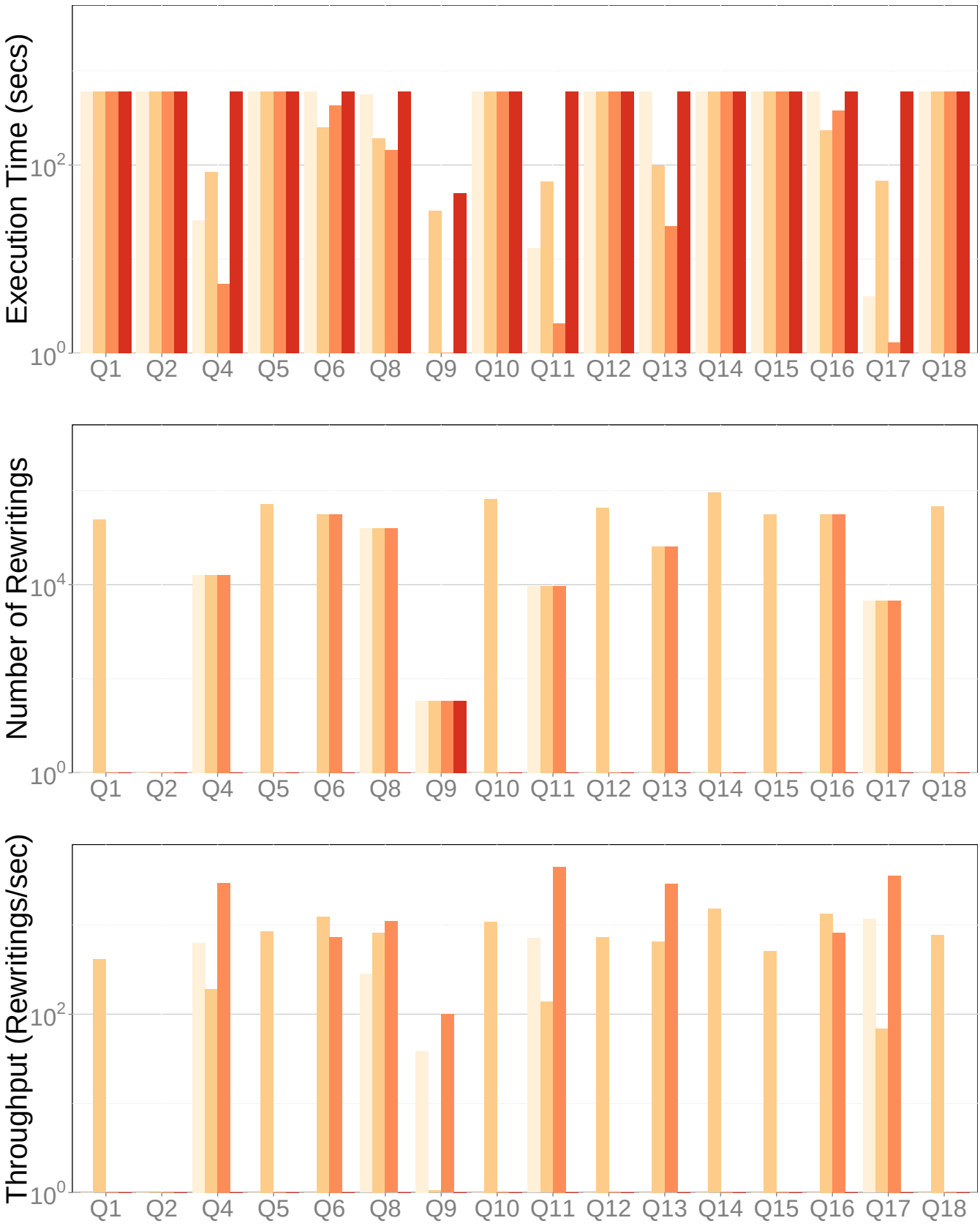


Figure 4.1 – Comparison of state-of-the-art LAV rewriting engines for 16 queries without existential variables and 476 views from our experimental setup. Studied engines are: GQR ( ), MCDSAT ( ), MiniCon ( ) and SSDSAT ( )

the defined views. The query triple patterns can be grouped into chained connected star-shaped subqueries, that have between one and twelve subgoals with only distinguished variables, i.e., queries are free of existential variable. We define five additional views to cover all the predicates in the queries. From these 14 views, we produce 476 views by horizontally partitioning each original view into 34 parts, such that each part produces 1/34 of the answers given by the original view.

Queries and views are described in Tables 4.3a and 4.3b. The size of the complete answer is computed by including all the views into an RDF-Store (Jena) and executing the queries against this centralized RDF dataset.

We implement wrappers as simple file readers. For executing rewritings, we use one named graph per subgoal as done in [Le+11]. The Jena 2.7.4 [1] library with main memory setup is used to store and query the graphs. The semLAV algorithms are implemented in Java, using different threads for bucket construction, view inclusion and query execution to improve performance. The implementation and all evaluation results are available in the project website [2].

Table 4.4 – The semLAV Effectiveness. For 10 minutes of execution, we report the number of relevant views included in the global schema instance, the number of covered rewritings and the achieved effectiveness. Also values for total number of views and rewritings are shown

Query	Included Views	# Relevant Views	# Covered rewritings	# Rewritings	Effectiveness
Q1	30	408	2.28E+06	2.04E+10	0.000112
Q2	194	408	2.05E+23	1.57E+24	0.130135
Q4	156	374	8.77E+03	1.62E+04	**0.542017**
Q5	52	374	3.13E+06	7.48E+07	0.041770
Q6	44	136	2.13E+04	3.14E+05	0.067728
Q8	81	136	9.36E+04	1.57E+05	**0.595588**
Q9	34	34	3.40E+01	3.40E+01	**1.000000**
Q10	88	408	3.20E+05	4.40E+06	0.072766
Q11	77	136	5.24E+03	9.25E+03	**0.566176**
Q12	238	408	7.70E+08	1.50E+09	**0.514286**
Q13	245	408	4.26E+04	6.47E+04	**0.657563**
Q14	46	272	1.22E+04	2.52E+06	0.004837
Q15	70	442	5.12E+08	2.04E+10	0.025144
Q16	82	136	1.90E+05	3.14E+05	**0.602941**
Q17	56	136	1.90E+03	4.62E+03	0.411765
Q18	23	374	2.80E+05	1.20E+09	0.000234

Experimental Results

The analysis of our results focus on three main aspects: the semLAV effectiveness, memory consumption and throughput.

To demonstrate the semLAV effectiveness, we execute semLAV with a timeout of 10 minutes. During this execution, the semLAV algorithms select and include a subset of the relevant views; this

1. http://jena.apache.org/
2. https://sites.google.com/site/semanticlav/

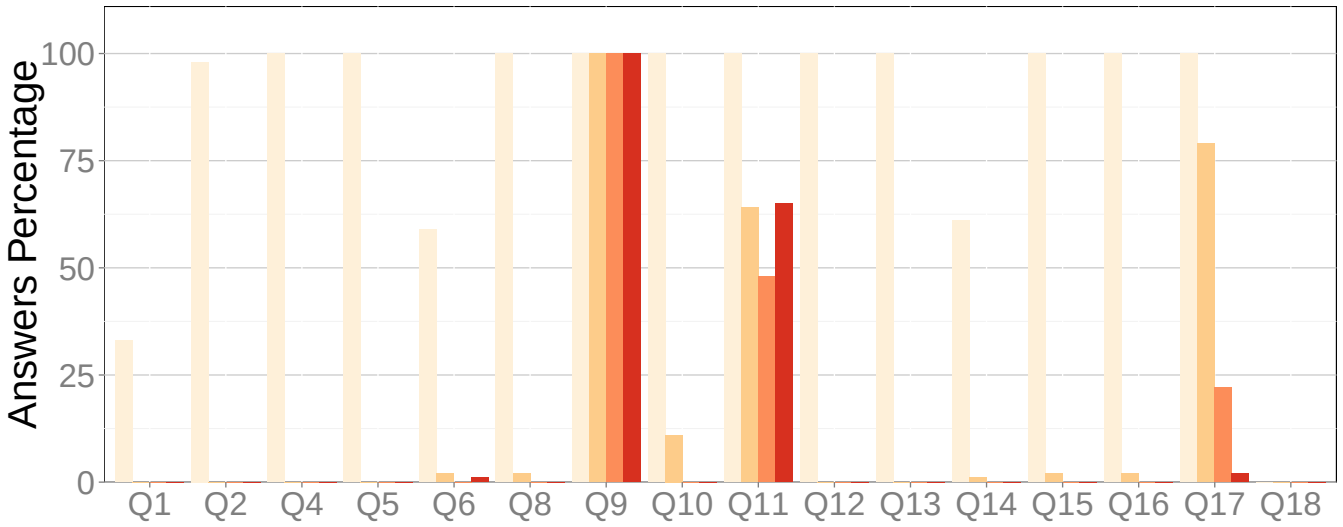


Figure 4.2 – Answer Percentage obtained by semLAV ( ), MCDSAT ( ), GQR ( ) and MiniCon ( )

set corresponds to V_k as a solution to the MaxCov problem. Then, we use these views to compute the number of covered rewritings using the formula given in Section 4.1. Table 4.4 shows the number of relevant views considered by semLAV, the covered rewritings and the achieved effectiveness. Effectiveness is greater than or equal to 0.5 (out of 1) for almost half of the queries. semLAV maximizes the number of covered rewritings by considering views that cover more subgoals first.

The observed results confirm that the semLAV effectiveness is considerably high. Effectiveness depends on the number of relevant views, but this number is bounded to the number of relevant views that can be stored in memory. As expected, the semLAV approach could require more space than the traditional rewriting-based approach. semLAV builds a global schema instance that includes all the relevant views in V_k, whereas a traditional rewriting-based approach includes only the views in one rewriting at the time.

We calculate the throughput as the number of answers divided by the total execution time. For semLAV, this time includes view selection and ranking, contacting data sources using the wrappers, including data into the global schema instance, and query execution time. For the traditional rewriting-based approach, this time includes rewriting time, instead of view selection and ranking.

Figures 4.2 and 4.3 show an impressive difference in the answer percentage and throughput, e.g., for Q1 semLAV produces 37,350.1 answers/sec, while the other approach produces up to 0.5 answers/sec. This huge difference is caused by the differences between the complexity of the rewriting generation and the semLAV view selection and ranking algorithm, and between the number of

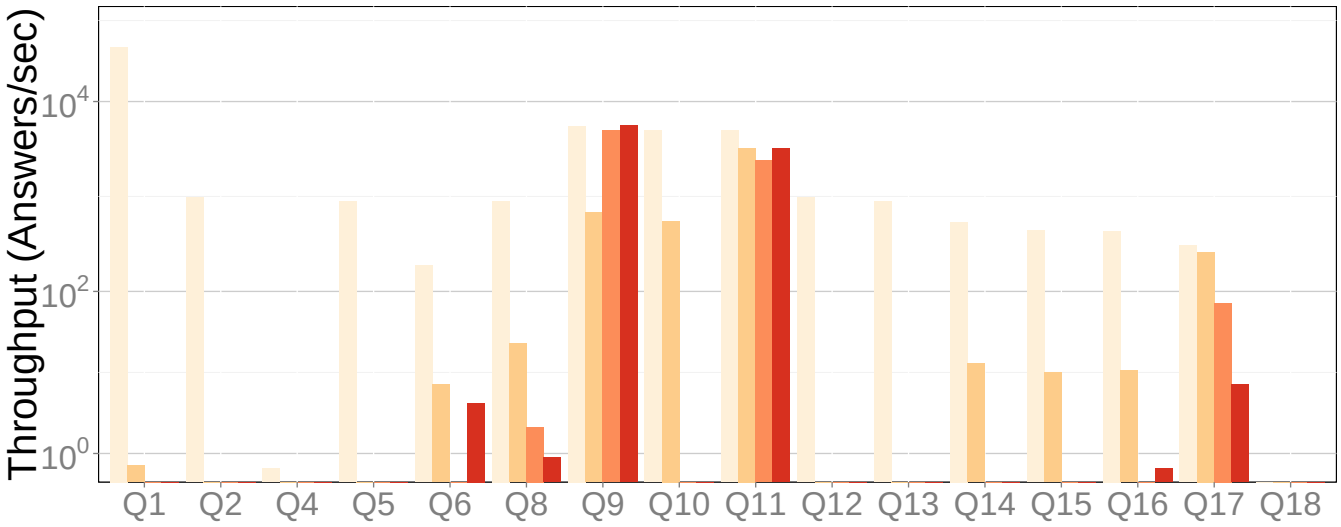


Figure 4.3 – Throughput of semLAV ( ), MCDSAT ( ), GQR ( ) and MiniCon ( )

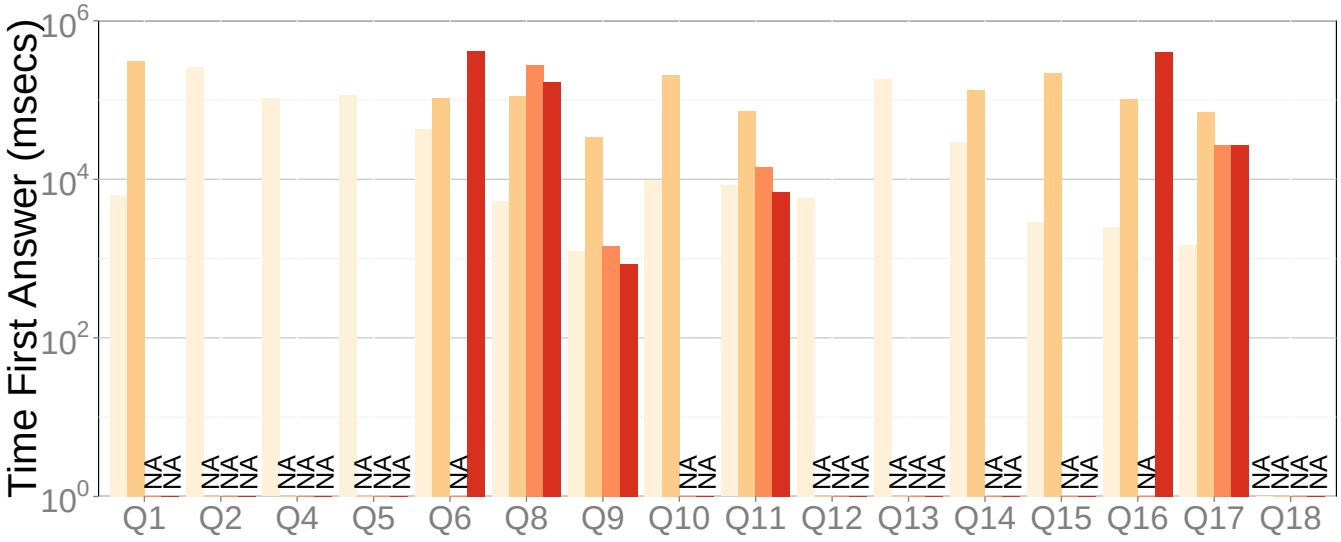


Figure 4.4 – Time of the First Answer (msec) of semLAV ( ), MCDSAT ( ), GQR ( ) and MiniCon ( ). "NA" indicates that the approach did not produce answers for that query

rewritings and number of relevant views. This makes possible to generate answers sooner.

Figure 4.4 shows the time for the first answer (TFA); TFA is impacted by executing the query as soon as possible, according to option 1 given in Algorithm 15. Only for query $Q18$ semLAV does not produce any answer in 10 minutes. This is because the views included in the global schema instance are large (around one million triples per view) and do not contribute to the answer; consequently, almost all the execution time is spent in transferring data from the relevant views. semLAV produces answers sooner in all the other cases. Moreover, semLAV also achieves complete answer in 11 of 16 queries in only 10 minutes.

Th results show that semLAV is effective and efficient and produces more answers sooner than

a traditional rewriting-based approach. semLAV makes the LAV approach feasible for processing SPARQL queries [Mon+13b; Mon+14a; Fol+14b; Fol+14a; Fol+15].

4.4 Conclusion

semlav In this chapter, I present semLAV, a Local-As-View mediation technique that allows to perform SPARQL queries over views without facing problems of NP-completeness, exponential number of rewritings or restriction to conjunctive SPARQL queries. This is obtained at the price of including relevant views into a global schema instance which is space consuming. However, I demonstrated that, even if only a subset of relevant views is included, I obtain more results than traditional rewriting-based techniques. Chances of producing results are higher, if the number of covered rewritings is maximized as defined in the MaxCov problem. We proved that our ranking strategy maximizes the number of covered rewritings.

semLAV opens a new way to execute SPARQL queries for LAV mediators that is tractable. As perspectives, the performance of semLAV can be greatly improved by parallelizing views inclusion. Currently, SemLAV includes views sequentially due to Jena restrictions. If views were included in parallel, time to get first results would be greatly improved. Additionally, the strategy of producing results as soon as possible, can deteriorate the overall throughput. If users want to improve overall throughput, then the query should be executed once after all the views in V_k have been included. It could be also interesting to design an execution strategy where SemLAV would execute under constrained space. In this case, the problem would be to find the minimum set of relevant views that would fit in the available space and produce the maximal number of answers. All these problems will be part of our future works.

The work presented in this chapter is part of Gabriela Montoya [Mon16] and Luis-Daniel Ibáñez[Ibá15] in collaboration with Maria-Esther Vidal, Professor at the University Simon Bolivar, Venezuela.

Publications associées à ce chapter

International peer-reviewed journals

[Mon+14a] Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « SemLAV: Local-as-View Mediation for SPARQL queries ». In: *LNCS Trans-*

actions on Large-Scale Data- and Knowledge-Centered Systems 8420 (2014), pp. 33–58.

International peer-reviewed conferences

[Mon+13b] Gabriela Montoya, Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « GUN: An Efficient Execution Strategy for Querying the Web of Data ». In: *Database and Expert Systems Applications*. Springer. 2013, pp. 180–194.

International peer-reviewed workshops

[Fol+15] Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « Parallel Data Loading during Querying Deep Web and Linked Open Data with SPARQL ». In: *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, PA, USA, October 11, 2015.* 2015, pp. 63–74.

5

Read/Write Linked Open Data

Motivations : The current web of Linked Data is Read-Only, therefore, when a data consumer detects inconsistencies, she cannot fix them in place. To be able to do so, the web of Linked Data needs to evolve from Read-Only to *Read/Write* [BO13]. If participants of the Web of Linked Data could write, data could be cleaned and evolve with the collaborative intervention of human and machine agents. The knowledge stored by different communities or even by different individuals or applications could *co-evolve* [EL88].

The paradigm of shifting from Read-Only to Read/Write would have a similar impact on the Web of Linked Data as the one of the advent of the social web had to the Web of Documents. The shift from the Web 1.0 to 2.0 allowed the collaborative edition of documents. The Web of Data could draw the same benefits to allow the collaborative construction of knowledge.

Contributions : Data consumers copy data from different sources in order to perform intensive querying, keeping themselves up-to-date through live update feeds or notification protocols. While querying, mistakes can be identified and repaired. These updates can be integrated by the sources or exchanged between data consumers through copying or through *pull requests*, in the spirit of Distributed Version Control Systems (DVCS).

Figure 5.1 illustrates this vision. The top three boxes represent three major Linked Data pub-

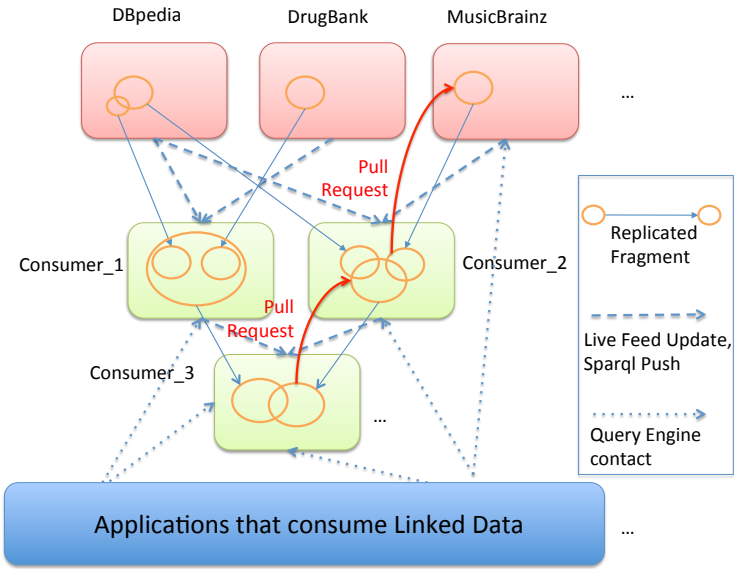


Figure 5.1 – Federation of Read/Write Linked Data

lishers, DBpedia [1], DrugBank [2] and MusicBrainz [3]. *Consumer*_1 copies fragments from DBPedia and DrugBank, *Consumer*_2 copies fragments from DBPedia and MusicBrainz and *Consumer*_3 copies fragments from *Consumer*_1 and *Consumer*_2. Applications that consume Linked Data can request data from consumers besides that from the original data publishers. The main contributions are: (1) SU-Set is a Conflict-Free Replicated Data Type (CRDT) for RDF Graph, it follows an optimistic replication model [SS05b]. SU-Set enables large scale replication of RDF graphs while ensuring strong eventual consistency, *i.e.*, when the system is idle all the replicas are convergent. SU-Set requires the connectivity of the network and the need to exchange all updates.

(2) Col-Graph is a coordination-free protocol based on annotated RDF-Graphs and updates to achieve fragments consistency. It follows a data sharing model [Kar+13]. Contrary to SU-Set, Col-Graph enables partial replication of RDF graph while ensuring incremental maintenance of the data fragment. Col-Graph is well adapted for socially generated networks.

1. http://dbpedia.org
2. www.drugbank.ca
3. http://musicbrainz.org

5.1 SU-Set: a Conflict-Free Replicated Data Type for RDF Graph

A Conflict-Free Replicated Data Type (CRDT) [Sha+11] is a data type whose operations, when concurrent, yield the same result regardless the execution order. Two operations are concurrent if they occur at different nodes and we cannot determine which one happened before the other. Common data types are in general not CRDTs, for example, the set data type is not a CRDT, because concurrent insertion and deletion does not commute.

A CRDT has three components :

— *payload* : the internal structure that holds the state of the object,

— *lookup* : queries the *payload* and returns data elements,

— *update*: handling operations such as *add* and *remove*.

An *update* operation is prepared at the generator site and sent to all nodes including the generator one :

— *prepare* : preparing the arguments for sending the operation to other nodes, if the preconditions of the operation do not hold, the operation is ignored,

— *effect* : sends the prepared operation to all nodes, including the generated one. When the operation is received, it will be executed if the preconditions are evaluated to true, else it will be delayed until they do so.

A CRDT ensures Strong Eventual Consistency (SEC) [SS05b; Sha+11] where *replicas that have delivered the same updates have equivalent state*.

I propose SU-Set a CRDT for the RDF-Graph type with SPARQL 1.1 Update operations that ensures SEC consistency for RDF Graph.

5.1.1 Algorithms

In the CRDT model operations are assumed to be transmitted through a fully connected communication graph without loss, granting the *Eventual Delivery* condition of SEC. Therefore, we introduce on the Read/Write Web of Linked Data the assumption that the update exchange network is strongly connected, all updates eventually arrive to their destination.

```
1  payload set S
2    initial ∅
3  query lookup (element e) : boolean b
4    let b = (∃u : (t, u) ∈ S)
5   update insert (set<element> T)
6    prepare(T)
7        let R = ∅
8        foreach t in T:
9            let α = unique()
10           R := R ∪ {(t, α)}
11   effect(R)
12       S := S ∪ R
```

Specification 5.1 – Union extension to OR-Set

SPARQL 1.1 Update graph update operations can be expressed as set union and difference on RDF-Graphs. Therefore, we can adapt the existing CRDTs for the set type. SU-Set extends Observed-Removed Set (OR-Set) [Sha+11] single-element insertion and deletion to union and difference. Specification 5.1, shows insert operation. Figure 5.2 shows a SU-Set execution, SPARQL 1.1 Update operations executed at Graph Stores are rewritten to SU-Set operations over pairs (triple,id) in a transparent way for the user, and sent downstream, where they are re-executed upon reception.

SU-Set inherits from OR-Set a precondition concerning the delivery of updates: *It must be granted that deletions of unique pairs are always delivered after the insertions that generated them.* We will use the same strategy used in OR-Set to make the pre-condition hold: a causal delivery of updates, implemented with *Vector Clocks* [Mat89]. Each Graph Store holds a monotonically increasing counter (the clock) that ticks each time an update is made, and an array whose keys are the identifiers of all other Graph Stores and whose values are the last received clock values from the corresponding Graph Store. Updates are piggybacked with the full vector at execution time. By comparing the vectors, one can determine the partial order of update executions in all the network to ensure that a deletion always happens after its corresponding insertion.

5.1.2 Evaluating and Optimizing SU-Set

This first version of SU-Set has two important overheads to consider. In the delete-insert operations, computing the triples affected locally and sending them downstream instead of sending the patterns directly greatly increases the traffic; second, if each element needs to be sent with its globally unique id, the size of the packets sent will grow.

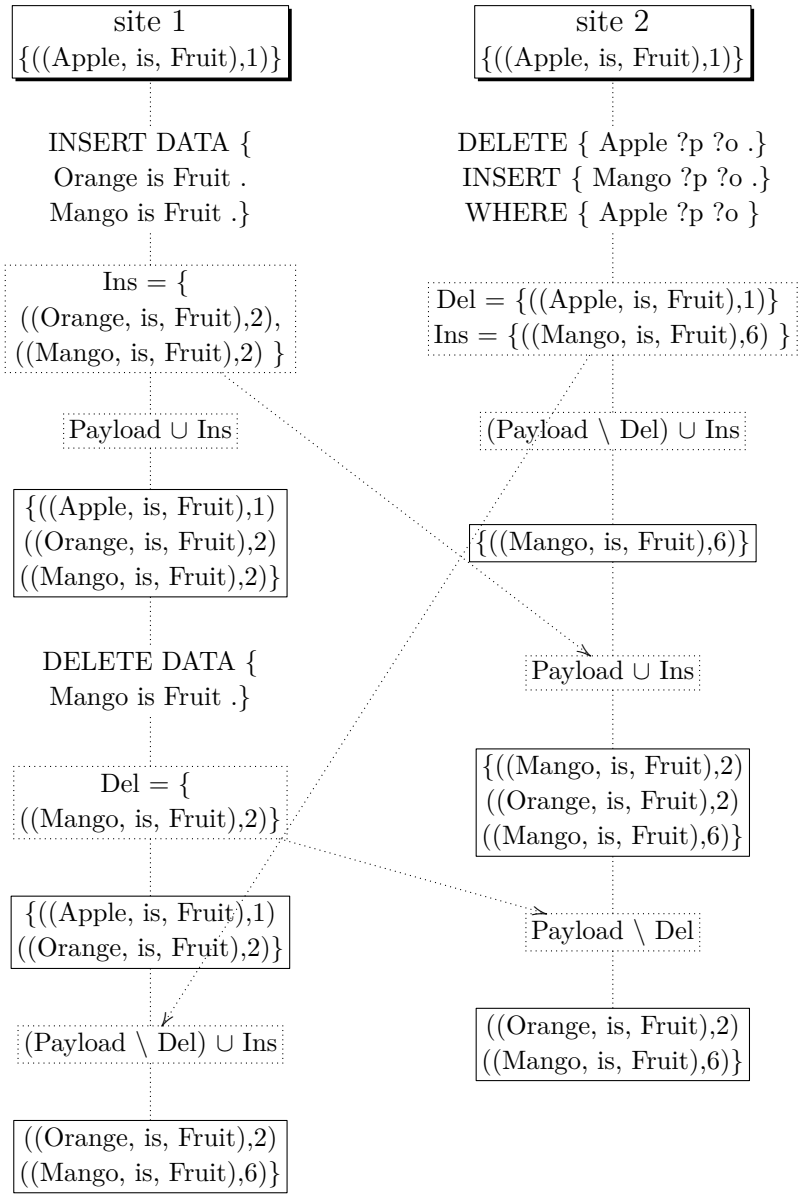


Figure 5.2 – SU-Set Execution

```
1  payload set S
2      initial ∅
3  query lookup (triple t) : boolean b
4      let b = (∃u : (t, u) ∈ S)
5  update insert (set<triple> T)
6      prepare(T)
7          let α = unique()
8      effect(T, α)
9          let R = {(t, α) : t ∈ T}
10         S := S ∪ R
11 update delete (set<triple> T)
12     prepare(T)
13         let R = ∅
14         foreach t in T:
15             let Q = {(t, u) | (∃u : | (t, u) ∈ S)}
16             R := R ∪ Q
17     effect(R)
18         // Causal Reception
19         pre All add(t,u) delivered
20         S := S \ R
21 update delete − insert(whrPat, delPat, insPat)
22     // match(m, pattern): triples matching
23     //     pattern within mapping m.
24     prepare(whrPat, delPat, insPat)
25         let S′ = {t | (∃u | : (t, u) ∈ S)}
26         // M is a Multiset of mappings
27         let M = eval(Select *
28                      from S′ where whrPat)
29         let D′ = ∅
30         foreach m in M:
31             D′ = D′ ∪ match(m, delPat)
32         let D = {(t, u) : t ∈ D′ ∧ (t, u) ∈ S}
33         let I′ = ∅
34         foreach m in M:
35             let I′ = I′ ∪ match(m, insPat)
36         let α = unique()
37     effect(D, I′, α)
38         // Causal Reception
39         pre All add(f,u) ∈ D delivered
40         let I = {(i, α) : i ∈ I′}
41         S := (S \ D) ∪ I
```

Specification 5.2 – Optimized SU-Set

To test this in a real case, we analyzed the publishing method of DBpedia Live. The core of the system is a set of extractors that computes the triples affected each time there is an edition in a Wikipedia page or in the mappings that define the relation between info boxes and triples. After updating the store, the system writes two files, one with the added RDF-triples, and another with the deleted ones. These files do not have a fixed size, as this depends on the number and nature of the editions at a given time. DBpedia Live publishing can be considered as SPARQL Update Insert Data and Delete Data operations with the triples defined by the change set files. This means that for the DBpedia Live case, the overhead of computing and sending the affected triples for each operation is already considered and SU-Set do not adds any further cost.

To evaluate the impact of using globally unique ids, we consider its implementation with two UUIDs [LMS05] of 16 bytes each, one to identify the generator site, and another to hold a big enough monotonic counter that increases with each insert. We downloaded the N3 files published by DBpedia Live from march 10th to march 16th 2012, totalizing 3,403 Megabytes, and appended to each triple the base64 representation of the two UUIDs. Finally, we measured the new file size with the UNIX command wc -c. The difference between the version with ids and the version without ids was 2,04 GB, and the percentage of increase, 54,47%.

Table 5.1 – Comparison of communication overhead between SU-Set, its optimized version and the use of no ids (nothing). The data used are the triples published by DBpedia Live from the 10th to the 16th march 2012.

		Size (MB)		
Operation	# Triples	Nothing	SU-Set	Optim.
Insert	21762190	3294,08	5334,29	3296,6
Delete	1755888	265,78	164,61 (430,4)	164,61 (430,4)
Total	23518078	3559,86	5498,9 (5794.69)	3461,21 (3727)
Overhead			54,47% (64,77%)	-2,77% (4,68%)

However, we can greatly reduce this overhead if the receivers can afford to spend some time in constructing the IDs from a resume. The strategy varies depending on the strategy chosen to achieve the delivery condition. When vector clocks are used, we showed in [Dan+12] that the same *id* can be shared by the triples inserted in the same operation, as the uniqueness of the element comprised by the triple and its id is maintained. Therefore, one can send only one *id* per insert operation and let the receiver reconstruct the pairs. Specification 5.2 details the optimized version of SU-Set.

In DBpedia Live, this would mean that only one *id* is needed for each file containing added triples.

We recomputed the overhead in our case of study using this strategy and we obtained a negligible 2,5 Megabytes for the insertions and a 4,68% file size increase overall. Note that, as the average triple size is greater than the *id* size, deletion with the first version of SU-Set is less expensive than without using any ids.Table 5.1 compares the differences in communication overhead of insertions and deletions between the two versions of SU-Set and the current publication method without ids. Note that in both solutions, it is possible to further optimize the deletion by sending only the id (as it is unique), however, in the case that we would like to analyze the deletes, an extra computation effort needs to be done to search the triples. As such, we report the overhead of sending only the ids and, in parentheses, the overhead of sending full pairs.

The poof of correctness and the complexity of algorithms are detailed in [Ibá+13].

5.2 Col-Graph: A Synchronization Algorithm for RDF Fragments

Imagine a participant that wants to perform data cleansing on a subset of DBpedia, e.g., the triples having as subject the entity *DBpedia:France*. Copying the entire DBpedia is a waste of resources, thus, the participant copies only the *fragment* of data she is interested to, and receives only the updates from DBpedia that concern such fragment. SU-Set cannot assert the consistency of such fragment, because both participants have not applied the same updates. In order to support this use case, we propose *Fragment Consistency*, a criterion focused on the consistency of the fragments of data copied instead of on the state equivalence of all participants

5.2.1 Fragment Consistency

Definition 5.2.1 (Fragment)**.** *Let S be a SPARQL endpoint of a participant, a fragment of the RDF-Graph made accessible by S, $F[S]$, is a SPARQL CONSTRUCT federated query where all graph patterns are contained in a single SERVICE block with S as the remote endpoint. We denote as $eval(F[S])$ the RDF-Graph result of the evaluation of $F[S]$.*

Definition 5.2.2. *Let S be an RDF-GraphStore, we say that an RDF-GraphStore T* Materializes a Fragment of *S, if T evaluates $F[S]$ and unions the result with its own data. The subset of updates made by S delivered to T are the ones that concern $F[S]$. We call S the* source *of the fragment and T its* target.

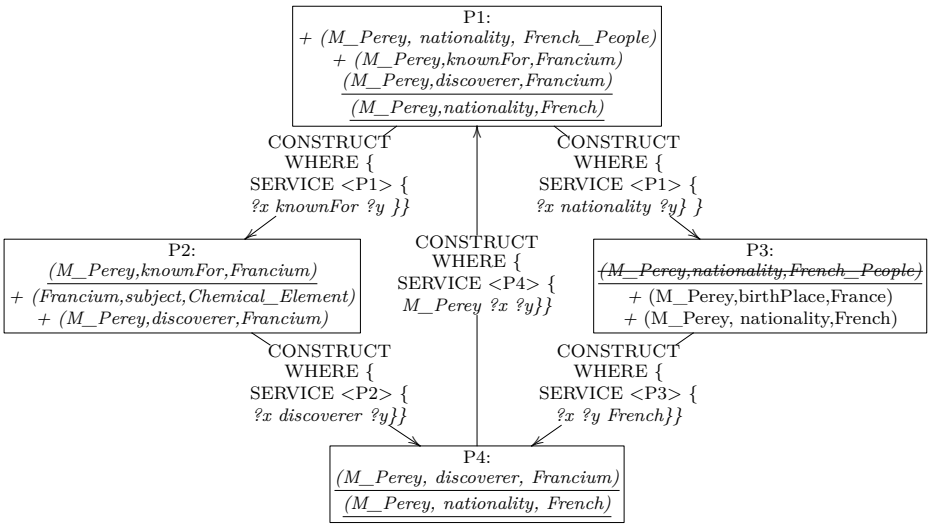
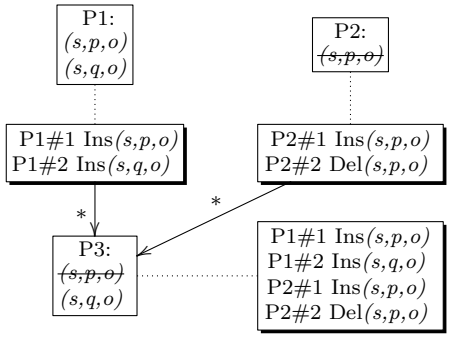


Figure 5.3 – Read/Write Linked Data with Fragments. Underlined triples are the ones coming from fragments, triples preceded by a '+' are the ones locally inserted, struck-through triples are the ones locally deleted.

Figure 5.3 illustrates how updates are propagated on Read/Write Linked Data using fragments. $P1$ starts with data about the *nationality* and *KnownFor* properties of M_Perey (prefixes are omitted for readability). $P2$ materializes from $P1$ all triples with the *knownFor* property. With this information and its current data, $P2$ inserts the fact that M_Perey discovered Francium. On the other hand, $P3$ materializes from $P1$ all triples with the *nationality* property. $P3$ detects a mistake (nationality should be $French$, not $French\_People$) and promptly corrects it. $P4$ constructed a dataset materializing from $P2$ the fragment of triples with the property *discoverer* the fragment of triples with the property *nationality* from $P3$. $P1$ trusts $P4$ about data related to M_Perey, so she materializes the relevant fragment, indirectly consuming updates done by $P2$ and $P3$.

Triples updated on materialized fragments are not necessarily integrated by the source, *e.g*, the deletion done by $P3$ did not reach $P1$, therefore, equivalence between source and materialized fragment cannot be used as consistency criterion. Intuitively, each materialized fragment must be equal to the evaluation of the fragment at the source after applying *local* updates, *i.e.*, the ones executed by the participant itself and the ones executed during synchronization with other fragments.

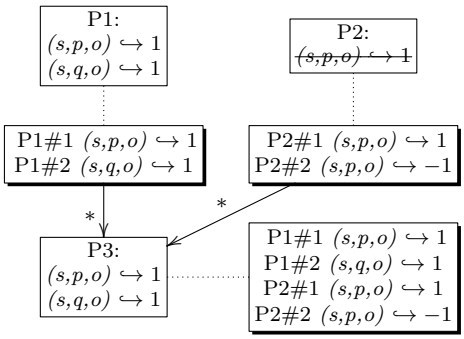
Definition 5.2.3 (Fragment Consistency)**.** *Let $RWLD = (P, E)$ be the Read/Write Linked Data. Assume each $P_i \in P$ maintains a sequence of uniquely identified updates Δ_{P_i} with its local updates and the updates it has consumed from the sources of the fragments $F[P_j]@P_i$ it materializes. Given a Δ_P, let $\Delta_P^{F[S]}$ be the ordered subset of Δ_P such that all updates* concern $F[S]$, i.e., *that match the graph pattern in $F[S]$. Let $apply(P_i, \Delta)$ be a function that applies a sequence of updates Δ on P_i.*



$F[P1]@P3 = apply(\{(s,p,o),(s,q,o)\}, (P2\#1\ Ins(s,p,o), P2\#2\ Del(s,p,o)))$

$F[P2]@P3 \neq apply(\emptyset, (P1\#1\ Ins(s,p,o)) P1\#2\ Ins(s,q,o)))$

(a) Applying updates as they come does not comply with the correction criterion.



$F[P1]@P3 = apply(\{(s,p,o) \hookrightarrow 1, (s,q,o) \hookrightarrow 1\}, (P2\#1\,(s,p,o) \hookrightarrow 1\ (P2\#2\,(s,p,o) \hookrightarrow -1))$

$F[P2]@P3 = apply(\emptyset, (P1\#1\,(s,p,o) \hookrightarrow 1, P1\#2\,(s,q,o) \hookrightarrow 1))$

(b) The Annotated RDF-Graph enables a consistent Collaboration Network

Figure 5.4 – Illustration of Fragment Consistency. Plain boxes represent RDF-Graphs, shaded boxes simplified sequences of updates. $*$ represents a full fragment.

RWLD is consistent *iff when the system is idle, i.e., no participant executes local updates or fragment synchronization, then:*

$$(\forall P_i, P_j \in P : F[P_i]@P_j = apply(eval(F[P_i]), \Delta_{P_j}^{F[P_i]} \setminus \Delta_{P_i})$$

The $\Delta_{P_j}^{F[P_i]} \setminus \Delta_{P_i}$ term formalises the intuition that we need to consider only local updates when evaluating the consistency of each fragment, *i.e.*, from the updates concerning the fragment, remove the ones coming from the source.

Unfortunately, applying remote operations as they come does not always comply with Definition 5.2.3 as shown in Figure 5.4a: $P3$ synchronizes with $P1$, applying the updates identified as $P1\#1$ and $P1\#2$, then with $P2$, applying the updates identified as $P2\#1$ and $P2\#2$, however, the fragment materialized from $P2$ is not consistent. Notice that, had $P3$ synchronized with $P2$ before than with $P1$, its final state would be different ((s,p,o) would exist) and the fragment materialized from $P1$ would not be consistent.

SU-Set algorithm 5.2 also cannot achieve Fragment Consistency due to its requirement of causal delivery of updates. Suppose a fragment of P_1 materialized at P_2 $F[P_1]@P_2$, and suppose that P_1 performs two updates, u_1 not concerning F and u_2 concerning F. In our model, u_1 will not be delivered to P_2, meaning that when u_2 is delivered, it will be considered as not causally ready to be executed and put on hold indefinitely.

5.2.2 Algorithms

To achieve Fragment Consistency, we propose, in the spirit of [GIT11], to count the number of insertions and deletions of a triple, *i.e.,* we annotate each RDF-triple with positive or negative integers, positive values indicate insertions and negative values deletions. This allows for a uniform representation of data and updates, yielding a simple way to synchronize fragments.

Incrementally synchronizing a materialized fragment using only the updates published by a data source and the locally materialized fragment without issuing another query on the data source requires to exclude join conditions from fragments [GJM96], therefore, to not compromise the availability of sources, we restrict to *basic fragments* [Ver+14b], *i.e.,* fragments where the query is comprised by only one triple pattern.

Definition 5.2.4 (Annotated RDF-triple, Graph and Update)**.** *1. Let t be an RDF-triple and $z \in \mathbb{Z}^*$. $t \hookrightarrow z$ is an* annotated RDF-triple*, t is called the* triple *and z the* annotation*.*

2. *An* annotated RDF-Graph G^A *is a set of annotated RDF-triples such that* $(\forall t, z | t \hookrightarrow z \in G^A : z > 0)$

3. *An* annotated update u^A *is represented by an annotated RDF-triple. More precisely, $t \hookrightarrow 1$ for insertion of t and $t \hookrightarrow -1$ for deletion of t.*

Annotations in RDF-Graphs count the number of *derivations* of a triple in the *RWLD*.

Definition 5.2.5 (Derivation)**.** *Let t be a triple stored in a participant RWLD P_0, a* derivation *of t is a simple path from the participant that inserted t, P_1, and P_0, such that the insertion of t concerns each edge of the path.*

An annotation value higher than 1 indicates that the triple exists in more than one source or there are several simple paths in *RWLD* leading from the participant that inserted the triple to the participant. Annotations in updates indicate, if positive, that z derivations of t were inserted; if negative, that z derivations of t were deleted. For example, an annotated RDF-triple $t_1 \hookrightarrow 2$ means that either t_1 has been inserted by two different sources or the same insert arrived through two different paths in *RWLD*. The annotated update $t_2 \hookrightarrow -1$ means that t_2 was deleted at one source or by some participant in the path between the source and the target; $t_3 \hookrightarrow -2$ means that either t_3 was deleted by two sources or by some participant in the path between two sources and the target.

To apply annotated updates to annotated RDF-Graphs, we define an *Update Integration* function:

Definition 5.2.6 (Update Integration)**.** *Let A be the set of all annotated RDF-Graphs and B the set of all annotated updates. Assume updates arrive and are executed from source to target in FIFO order. The* Update Integration *function $\uplus : A \times B \to A$ takes an annotated RDF-Graph $G^A \in A$ and an annotated update $t \hookrightarrow z \in B$:*

$$G^A \uplus t \hookrightarrow z = \begin{cases} G^A \cup \{t \hookrightarrow z\} & if\ (\nexists w : t \hookrightarrow w \in G^A) \\ G^A \setminus \{t \hookrightarrow w\} & if\ t \hookrightarrow w \in G^A \wedge w + z \leq 0 \\ (G^A \setminus \{t \hookrightarrow w\}) \cup \{t \hookrightarrow w + z\} & if\ t \hookrightarrow w \in G^A \wedge w + z > 0 \end{cases}$$

The first piece of the Update Integration function handles incoming updates of triples that are not in the current state. As we are assuming FIFO in the update propagation from source to target, insertions always arrive before corresponding deletions, therefore, this case only handles insertions. The second piece handles deletions: only if the incoming deletion makes the annotation zero the triple is deleted from the current state. The third piece handles deletions that do not make the annotation zero and insertions of already existing triples by simply updating the annotation value.

We now consider each participant has an annotated RDF-Graph G^A and an sequence of annotated updates U^A. SPARQL queries are evaluated on the RDF-Graph $\{t \,|\, t \hookrightarrow z \in G^A\}$. SPARQL Updates are also evaluated this way, but their effect is translated to annotated RDF-Graphs as follows: the insertion of t to the insertion of $t \hookrightarrow 1$ and the deletion of t to the deletion of the annotated triple having t as first coordinate. Specification 5.3 details the methods to insert/delete triples and synchronize materialized fragments. To avoid the infinite forwarding of updates, each time an update is processed, the protocol checks if it has walked a cycle, if so, it is ignored. Figure 5.4b shows the fragment synchronization algorithm in action.

To materialize fragments for the first time, a SPARQL extension that allows to query the annotated RDF-Graph and return the triples *and* their annotations is needed, for example the one implemented in [WCG14]. To check when an update has cycled, we propose to add a second annotation to updates, containing a set of participant identifiers ϕ_u representing the participants that have already received and applied the update. When an update u is created, ϕ_u is set to the singleton containing the ID of the author, when u is pushed downstream, the receiving participant checks if

```
1  Annotated Graph G^A,
2  Sequence ΔP_ID
3
4  void insert(t):
5      pre: t ∉ {t'|t ↪ x ∈ G^A}
6      G^A := G^A ∪ t ↪ 1
7      Append(ΔP_ID, t ↪ 1)
8
9  void delete(t):
10     pre: t ∈ {t'|t' ↪ x ∈ G^A}
11     G^A := G^A ⊎ t ↪ −z
12     Append(ΔP_ID, t ↪ −z)
13
14 void sync(F[P_x], ΔP_x):
15     for t ↪ z ∈ ΔP_x:
16         if t ↪ z has not cycled:
17             G^A := G^A ⊎ t ↪ z
18             Append(ΔP_ID, t ↪ z)
```

Specification 5.3 – Class Participant when triples are annotated with elements of Z.

```
1  IRI P_ID,
2  Annotated Graph G^A,
3  Sequence ΔP_ID
4
5  void insert(t):
6      pre: t ∉ {t'|t ↪ x ∈ G^A}
7      G^A := G^A ∪ t ↪ P_ID
8      Append(ΔP_ID, t ↪ P_ID)
9
10 void delete(t):
11     pre: t ∈ {t'|t' ↪ x ∈ G^A}
12     G^A := G^A ⊎ t ↪ −m
13     Append(ΔP_ID, t ↪ −m)
14
15 void sync(F[P_x], ΔP_x):
16     for t ↪ m ∈ ΔP_x :
17         if t ↪ m has not cycled:
18             G^A := G^A ⊎ t ↪ m
19             Append(ΔP_ID, t ↪ m)
```

Specification 5.4 – Class Participant when triples are annotated with elements of the monoid M.

his ID is in ϕ_u, if yes, u has already been received and is ignored, else, it is integrated, and before pushing it downstream it adds its ID to ϕ_u. Of course, there is a price to pay in traffic, as the use of ϕ increases the size of the update. The length of ϕ_u is bounded by the length of the longest simple path in the Collaboration-Network, which in turn is bounded by the number of participants.

Fortunately, the issue described in the previous section can be solved if we make the deletions stop when they do not affect the current state instead of when cycles are detected, in a similar way to the fixepoint semantics of datalog. Specification 5.5 shows the modified version of the algorithm. Figure 5.5 illustrates how this versions fixes the problem. The core of the fix lies on the *sync* procedure, the check for cyclic updates is only done for insertions (line 17). For deletions the stop condition is that the triple is not anymore there.

Provenance for Conflict Resolution

In section 5.2.2 we solved the problem of consistent synchronization of basic fragments. However, Fragment Consistency is based on the mere existence of triples, instead of on the possible conflicts between triples coming from different fragments and the ones locally inserted. Col-Graph's strategy in this case is that each participant is responsible for checking the semantic correctness of its dataset,

```
1  Annotated Graph G^A,
2  Sequence ΔP_ID
3
4  void insert(t):
5      pre: t ∉ {t'|t ↪ x ∈ G^A}
6      G^A := G^A ∪ t ↪ 1
7      Append(ΔP_ID, t ↪ 1)
8
9  void delete(t):
10     pre: t ∈ {t'|t' ↪ x ∈ G^A}
11     G^A := G^A ⊎ t ↪ −z
12     Append(ΔP_ID, t ↪ −z)
13
14 void sync(F[P_x], ΔP_x):
15     for t ↪ z ∈ ΔP_x:
16         if z > 0:
17             if t ↪ z has not cycled:
18                 G^A := G^A ⊎ t ↪ z
19                 Append(ΔP_ID, t ↪ z)
20         if z < 0:
21             if G^A ⊎ t ↪ z ≠ G^A:
22                 G^A := G^A ⊎ t ↪ z
23                 Append(ΔP_ID, t ↪ z)
```

Specification 5.5 – Class Participant with Z annotations modified for all topologies

```
1  IRI P_ID,
2  Annotated Graph G^A,
3  Sequence ΔP_ID
4
5  void insert(t):
6      pre: t ∉ {t'|t ↪ x ∈ G^A}
7      G^A := G^A ∪ t ↪ P_ID
8      Append(ΔP_ID, t ↪ P_ID)
9
10 void delete(t):
11     pre: t ∈ {t'|t' ↪ x ∈ G^A}
12     G^A := G^A ⊎ t ↪ −m
13     Append(ΔP_ID, t ↪ −m)
14
15 void sync(F[P_x], ΔP_x):
16     for t ↪ m ∈ ΔP_x :
17         if m > 0:
18             if t ↪ m has not cycled:
19                 G^A := G^A ⊎ t ↪ m
20                 Append(ΔP_ID, t ↪ m)
21         if m < 0:
22             if G^A ⊎ t ↪ m ≠ G^A:
23                 G^A := G^A ⊎ t ↪ m
24                 Append(ΔP_ID, t ↪ m)
```

Specification 5.6 – Class participant with M annotations modified for all topologies

as criteria often varies and what is semantically wrong for one participant, could be right for another. Participants can delete/insert triples to fix what they consider wrong. Participants that receive these updates can edit in turn if they do not agree with them.

In the event that a participant wants to choose between two triples, the main criteria to choose which one of them delete is their *provenance*. With this information, the decision can be made based on the trust on their authors. As in [Kar+13], we propose to substitute the integer annotations of the triple by an element of a commutative monoid that embeds $(Z, +, 0)$.

Definition 5.2.7 (Commutative Monoid)**.** *A Commutative Monoid is is an algebraic structure comprised by a set K, a binary, associative, commutative operation $\oplus$ and an identity element $0_K \in K$ such that*

$$(\forall k \in K \mid k \oplus 0_K = k)$$

Definition 5.2.8 (Embedding)**.** *A monoid $M = (K, \oplus, 0_K)$ embeds another monoid $M' = (K', \oslash, 0_{K'})$ iff there is a map $f : K \to K'$ called* homomorphism *such that $f(0_K) = f(0_{K'})$ and $(\forall a, b \in K :$*

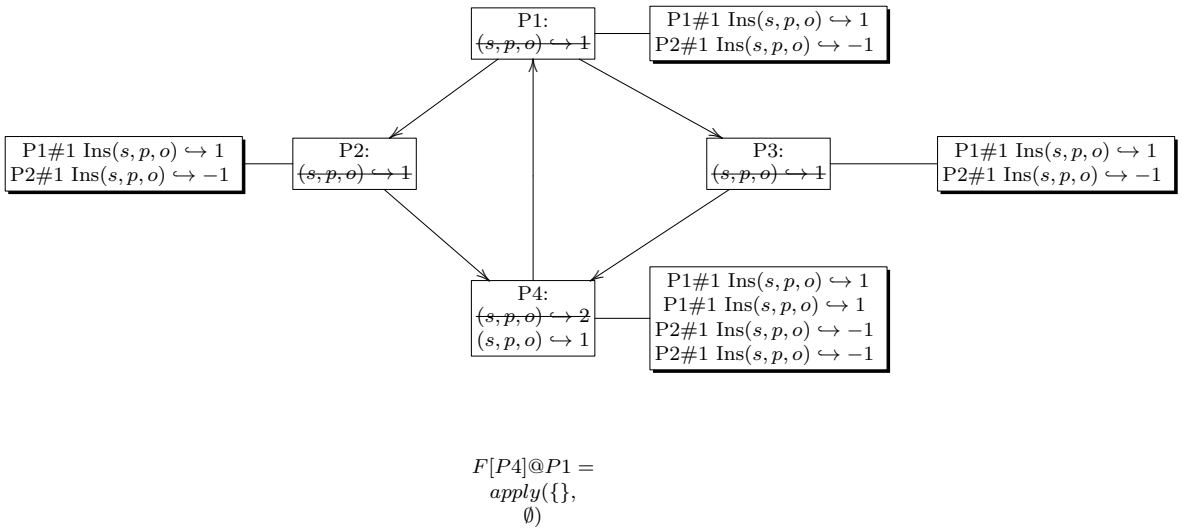


Figure 5.5 – Iterating deletions until no effect allows support for any network topology

$f(a \oplus b) = f(a) \oslash f(b)$).

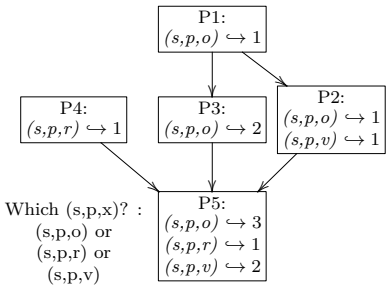
If we annotate with elements of a monoid that embeds $(Z, +, 0)$, all the properties of our synchronization algorithm maintain. Formally, the semantics of the querying commutes with the application of the homomorphism, a fundamental theorem proved in [GKT07] for the more general case of rings instead of monoids. The use of symbolic expressions that can be morphed to the basic $(Z, +, 0)$ allows the encoding of useful information, for instance, the provenance of triples.

Definition 5.2.9. *Assume each participant in the RWLD has an unique ID, and let X be the set of all IDs. Let $M = (Z[X], \oplus, 0)$ be a monoid with:*

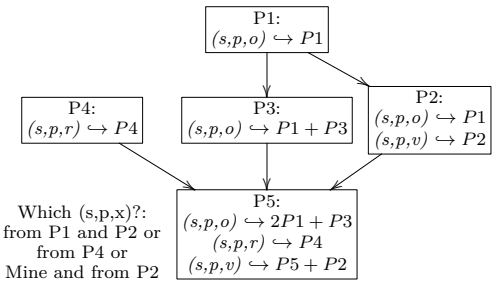
1. *The identity* 0.
2. *The set $Z[X]$ of polynomials with coefficients in Z and variable in X.*
3. *The polynomial sum $\oplus$, for each monomial with the same indeterminate: $aX \oplus bX = (a+b)X$*
4. *M embeds $(Z, +, 0)$ through the function $f(a_1X_1 \oplus \cdots \oplus a_nX_n) = \sum_1^n a_i$*

Each time a participant inserts a triple, she annotates it with its ID with coefficient 1. The only change in definition 5.2.6 is the use of $\oplus$ instead of $+$. Specifications 5.4 and 5.6 describes the algorithm to insert/delete triples and synchronize fragments with triples annotated with elements of M.

When annotating with Z, the only information encoded in triples is their number of derivations. M adds (i) Which participant is the *author* of the triple. A triple stored by a participant P with an annotation comprised of the sum of n monomials indicates that the triple was inserted *concurrently*



(a) Without provenance, P5 only information is the number of derivations. She does not know the author of the facts.



(b) With provenance, P5 also knows who inserted what and if it was concurrent, enabling trust based decisions to solve conflicts.

Figure 5.6 – Difference between annotating with Z (5.6a) versus annotating with M (5.6b).

by n participants from which there is a path in CN to P. (ii) The number of simple paths in the Collaboration Network in which all edges concern the triple, starting from the author(s) of the triple to this participant, indicated by the coefficient of the author's ID.

Figure 5.6 compares annotations with Z versus annotations with M. In the depicted collaboration network, the fact *(s,p,o)* is inserted concurrently by P1 and P3, *(s,p,v)* is inserted concurrently by P2 and P5 and *(s,p,r)* inserted only by P4. When the synchronization is finished, P5 notices that it has three triples with s and p as subject and predicate but different object values. If P5 wants to keep only one of such triples based on trust, the Z annotations (5.6a) do not give enough information, while the M annotations (5.6b) give more information for $P5$ to take the right decision. She can know that the triple (s, p, o) was inserted by two participants $P1$ and $P3$, while (s, p, r) was only inserted by $P4$ and that (s, p, v) was inserted by $P2$ and $P5$.

Col-Graph's performance is mainly affected by the following properties of the RWLD:

— The probability of concurrent insertion of the same data by many participants. The higher this probability, the number of terms of the polynomials is potentially higher.

— Its *connectivity*. The more connected, the more paths between the participants and the potential values of ρ are higher. If the network is poorly connected, few updates will be consumed and the effects of concurrent insertion are minimized.

— The *overlapping* between fragments. If all fragments copy all data, all incoming updates will be integrated by every participant, maximizing the effects of connectivity and concurrent insertion. If all fragments are disjoint, then all updates will be integrated only once and the effects of connectivity and concurrent insertion will be neutralized.

Details of complexity analysis and experimentations are given in [Ibá+14].

5.3 Conclusion

In this chapter, I proposed two solutions: SU-Set and Col-Graph to transform Read-Only LOD into Read/Write LOD. SU-Set's guarantees that : *Participants that have received the same updates, have the same state.* SU-Set has linear complexity in time and space independently of the topology, and optimal cost in communication. SU-Set is tailored for full dataset replication.

I proposed Col-Graph to support partial data replication. Each participant can copy or materialize from other participants a fragment of data defined by a SPARQL CONSTRUCT Federated query and receives the updates that concern the fragment. Col-Graph's guarantee that : *each materialized fragment is equal to the evaluation of the fragment at its source modulo the locally made updates.* Col-Graph has the same complexity of SU-Set's except in two aspects: (i) In space, where it depends on the connectivity of the network with a worst case of factorial (complete graph). (ii) In number of total messages exchanged in the network to converge. Nevertheless, our experiments suggest that for social networks, the performance is much better than for random networks, meaning that Col-Graph is applicable for the Read/Write Linked Data.

An interesting conclusion of this work is the very close relation between the solutions used for two very different visions of consistency by different research communities: Conflict-Free Replicated Data Types (CRDTs) in distributed systems and Collaborative Data Sharing Systems (CDSSs) in databases. Both resort to annotate data with elements of an algebraic structure, lattices in the case of CRDTs, and commutative semirings in the case of CDSSs. The main difference is the idempotency of lattices and the non-idempotency of semi-rings. Idempotency is required in replication scenarios to tolerate network disorder, on the other hand, semi-rings are required to support relational algebra operators but this feature requires coordination in the update exchange. We showed that for the special case of fragments, we can use semi-rings and still have the coordination freeness of idempotent solutions.

Finally, the connection between provenance and consistency maintenance is also worth to highlight. The data annotations used to model concurrency in CRDTs equal to one of the basic types of provenance semi-rings, therefore, if provenance information about triples is maintained in a semi-ring transformable format, then the consistency criteria proposed in this thesis can be attained.

Supervised PhD Thesis associated with this chapter

[Ibá15] Luis-Daniel Ibáñez. « Towards a Read/Write Web of Linked Data ». PhD thesis. Université de Nantes, Feb. 2015.

Publications associated with this chapter

International peer-reviewed journals

[Ibá+13] Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. « Live Linked Data: Synchronizing Semantic Stores with Commutative Replicated Data Types ». In: *International Journal of Metadata, Semantics and Ontologies* 8.2 (2013), pp. 119–133. URL: http://hal.inria.fr/hal-00903377.

International peer-reviewed conferences

[Ibá+14] Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. « Col-Graph: Towards Writable and Scalable Linked Open Data ». In: *ISWC - The 13th International Semantic Web Conference*. Riva del Garda, Italy, Oct. 2014, pp. 325–340.

[ASM12] Khaled Aslan, Hala Skaf-Molli, and Pascal Molli. « Connecting Distributed Version Control Systems Communities to Linked Open Data ». In: *CTS 2012 - The International Conference on Collaboration Technologies and Systems - 2012*. 2012.

[Asl+11a] Khaled Aslan, Nagham Alhadad, Hala Skaf-Molli, and Pascal Molli. « SCHO: An Ontology Based Model for Computing Divergence Awareness in Distributed Collaborative Systems ». In: *European Conference on Computer-Supported Cooperative Work*. Aarhus, Danemark, Sept. 2011.

[ASM10] Khaled Aslan, Hala Skaf-Molli, and Pascal Molli. « From Causal History to Social Network in Distributed Social Semantic Software ». In: *Web Science Conference 2010 - WebSci10*. Apr. 2010.

International peer-reviewed workshops

[Dan+12] Ibáñez Luis Daniel, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. « Synchronizing Semantic Stores with Commutative Replicated Data Types ». In: *SWCS - Semantic*

Web Collaborative Spaces Workshop - 2012. Lyon, France: ACM, 2012, pp. 1091–1096. URL: http://hal.inria.fr/hal-00686484.

[Ska+12] Hala Skaf-Molli, Emmanuel Desmontils, Emmanuel Nauer, Gérôme Canals, Amélie Cordier, Marie Lefevre, Pascal Molli, and Yannick Toussaint. « Knowledge Continuous Integration Process (K-CIP) ». In: *WWW 2012 - SWCS'12 Workshop - 21st World Wide Web Conference - Semantic Web Collaborative Spaces workshop*. Lyon, France, Apr. 2012, pp. 1075–1082. URL: http://hal.inria.fr/hal-00765596.

[Asl+11b] Khaled Aslan, Pascal Molli, Hala Skaf-Molli, and Stéphane Weiss. « C-Set : a Commutative Replicated Data Type for Semantic Stores ». In: *RED: Fourth International Workshop on REsource Discovery at 8th Extended Semantic Web Conference, ESWC 2011*. Heraklion, Grèce, May 2011.

6

Federated SPARQL Queries Processing with Replicated Fragments

Motivations: Existing SPARQL federated query engines do not support replicated data. In presence of replicated data, the performance of the state of the art federated query engines FedX [Sch+11] and ANAPSID [Aco+11] is degraded. Without the knowledge about replicated data, these engines may retrieve data from every relevant server, and produce a large number of intermediate results. Therefore, federated query engines may exhibit poor *performance* while *availability* of the selected SPARQL servers is negatively impacted. To illustrate, we replicated the DBpedia dataset [1] and defined two federations. The first one is composed of one mirror of DBpedia, and the second of two identical mirrors of DBpedia. We used FedX [Sch+11] and ANAPSID [Aco+11] federated query engines to execute the query in Figure 6.1a against both federations. In the first federation, both engines produced all the query answers in less than 23 seconds. For the second federation, the query engine has to contact both data sources because it has no knowledge about the relationship between data sources. Therefore, the performance in terms of execution time and number of transferred tuples, is seriously degraded as depicted in Figure 6.1b. For both engines the execution time and

1. DBpedia Live at August 15th, 2013.

```
select distinct ?p ?m ?n ?d where {
  ?p  dbprop:name ?m .
  ?p  dbprop:nationality ?n .
  ?p  dbprop:doctoralAdvisor ?d
}
```

(a) DBpedia Query

#DBpedia Replicas	FedX		ANAPSID	
	ET (s)	NTT	ET (s)	NTT
1	4.80	8,230	2.61	8,229
2	2,678.10	2,260,006	3,415.24	8,337,702

(b) Query Execution

Figure 6.1 – DBpedia query and its Execution Time (ET) and Number of Transferred Tuples (NTT) during query execution against federations with one and two replicas of DBpedia

number of transferred tuples increase more than 250 times when a second replica of DBpedia is added to the federation. Both query engines have to retrieve twice all the triples that match each of the triple patterns of the query, instead of evaluating the joins in the endpoints and retrieving only the query answers. For example, for the first triple pattern, the number of triples is greater than 4 millions. This number is likely to be higher than the maximum number of result rows that the endpoint is allowed to send, in consequence it risks to produce incomplete answers.

As the number of transferred tuples increases, the availability of the contacted SPARQL endpoints can be affected. *A replication aware federated query engine could select the SPARQL endpoints to contact in order to produce a complete query answer and transfer the minimum amount of data.*

Contributions: We build the first replication-aware SPARQL federated query engine by integrating into state-of-the art federated query engines FedX [Sch+11] and ANAPSID [Aco+11], a source selection strategy called Fedra that solves the source selection problem with fragment replication (SSP-FR). For a given set of SPARQL endpoints with replicated fragments and a SPARQL query, the problem is to minimize the number of transferred data from endpoints to the federated query engines, while preserving answer completeness and reducing data redundancy.

6.1 Problem Statement

Given a SPARQL query Q, a set of SPARQL endpoints E, the set of fragments F that have been replicated by at least one endpoint in E, a fragment mapping *endpoints()*, a containment mapping $\sqsubseteq$. The Source Selection Problem with Fragment Replication (SSP-FR) is to assign to each triple pattern in Q, the set of endpoints from E that need to be contacted to answer Q. A solution of SSP-FR corresponds to a mapping D that satisfies the following properties:

1. **Answer completeness:** sources selected in D lead engines to produce complete query answers.
2. **Data redundancy minimization:** *cardinality(D(tp))* is minimized for all triple pattern *tp* in Q, i.e., redundant data is minimized.

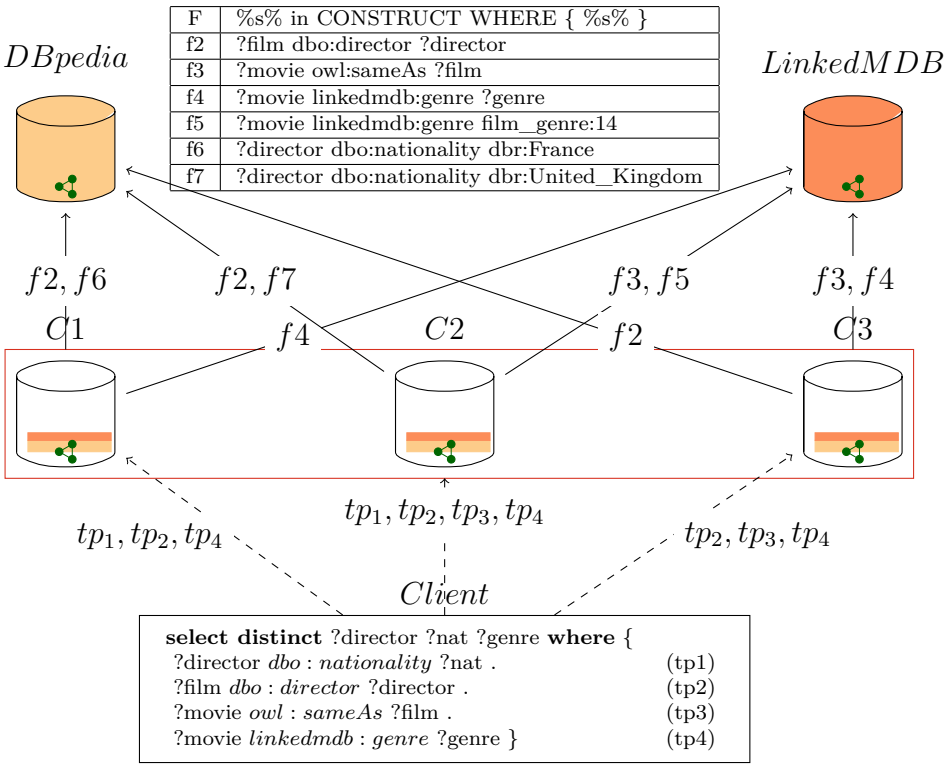


Figure 6.2 – Client defines a federation composed of $C1$,$C2$, and $C3$ that replicates fragments $f2-f7$

Table 6.1 – Q Relevant fragments, and source selections that lead to produce all the obtainable answers for the federation given in Figure 6.2

(a) Relevant Fragments

Q triple pattern		RF	Endpoints
tp1	?director dbo:nationality ?nat	f6	C1
		f7	C2
tp2	?film dbo:director ?director	f2	C1,C2,C3
tp3	?movie owl:sameAs ?film	f3	C2,C3
tp4	?movie linkedmdb:genre ?genre	f4	C1,C3
		f5	C2

(b) Source selections

TP	D_0(tp)	D_1(tp)	D_2(tp)
tp1	{C1,C2}	{C1,C2}	{C1,C2}
tp2	{C1,C2,C3}	{C1}	{C3}
tp3	{C2,C3}	{C2}	{C3}
tp4	{C1,C2,C3}	{C3}	{C3}
Tuples to transfer	421,675	170,078	8,953

3. **Data transfer minimization:** executing the query using the sources selected in D minimizes the number of transferred data.

We illustrate SSP-FR on running query Q of Figure 6.2. Table 6.1a presents relevant fragments for each triple pattern. Table 6.1b shows three $D(tp)$ that ensure the answer completeness property.

It may seem counterintuitive that these three *D(tp)* do ensure the answer completeness property, as they do not include existing DBpedia triples for *dbo:nationality* predicate with object different from *dbr:France* and *dbr:United_Kingdom*, but as they are not included in endpoints in E, these triples are inaccessible to the federation. Even if D_1 and D_2 minimize the number of selected endpoints per triple pattern, only D_2 minimizes the transferred data. Indeed, executing $tp1, tp2, tp3$ against replicated fragments that are located in the same data consumer endpoint will greatly reduce the

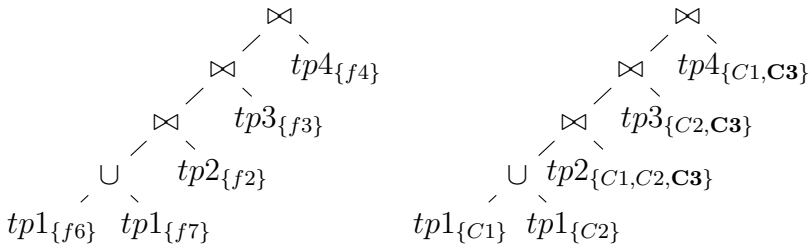


Figure 6.3 – Execution plan encoded in data structures R (left) and E (right); multiple subsets represent union of different fragment (ex. {f6}, {f7}); elements of the subset represent alternative location of fragments (ex. {C1,C3}); bold sources are the selected sources after set covering is used to reduce number of selected sources

size of intermediate results.

6.2 Algorithms

The goal of Fedra is to reduce data transfer by taking advantage of the replication of relevant fragments for several triple patterns on the same endpoint. Algorithm 16 proceeds in four main steps:

I. Identify relevant fragments for triple patterns, a Basic Graph Pattern (BGP) triple pattern can be contained in one fragment or a union of fragments (lines 5-6).

II. Localize relevant replicated fragments on the endpoints (line 7).

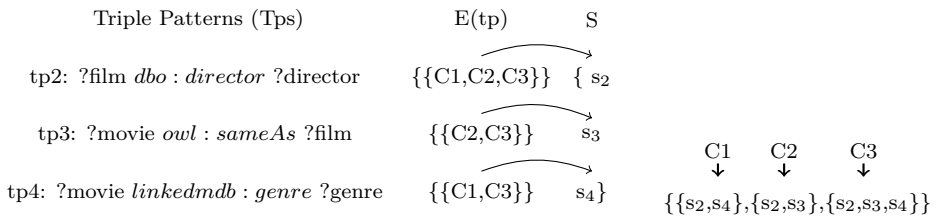
III. Prune endpoints for the unions (line 11).

IV. Prune endpoints for the BGPs using a set covering heuristic (line 12).

Algorithm 16 Fedra Source Selection algorithm

Require: Q: SPARQL Query; F: set of Fragments; endpoints : Fragment → set of Endpoint; $\sqsubseteq$: TriplePattern × TriplePattern
Ensure: selectedEndpoints: map from TriplePattern to set of Endpoint.

1: **function** SOURCESELECTION(Q,F,endpoints,$\sqsubseteq$)
2: triplePatterns ← get triple patterns in Q
3: R, E ← ∅, ∅
4: **for each** tp ∈ triplePatterns **do**
5: R(tp) ← RELEVANTFRAGMENTS(tp, F) ▷ Relevant fragments as in
6: R(tp) ← $\{\{f : f \in R(tp) : tp \sqsubseteq f\}\} \bigcup \{\{f\} : f \in R(tp) : f \sqsubseteq tp \wedge \neg(\exists g : g \in R(tp) : f \sqsubset g \sqsubseteq tp)\}$
7: E(tp) ← { ($\bigcup$ endpoints(f) : f ∈ fs) : fs ∈ R(tp) }
8: **end for**
9: basicGP ← get basic graph patterns in Q
10: **for each** bgp ∈ basicGP **do**
11: UNIONREDUCTION(bgp, E) ▷ endpoints reduction for multiple fragments triples
12: BGPREDUCTION(bgp, E) ▷ endpoints reduction for the bgp triples
13: **end for**
14: **for each** (tp, E(tp)) ∈ E **do**
15: selectedEndpoints(tp) ← for each set in E(tp) include one element
16: **end for**
17: **return** selectedEndpoints
18: **end function**



(a) S instances (b) C instance

Figure 6.4 – Set covering instances of S and C of BGP reduction Algorithm 18 for the query Q (Figure 6.2)

Next, we illustrate how Algorithm 16 works on our running query Q and data consumer endpoints $C1, C2, C3$ from Figure 6.2. [2]

First, for each triple pattern, Fedra computes relevant fragments in *R(tp)*, and groups them if they provide the same relevant data. For tp1, $R(tp1) \rightarrow \{\{f6\}, \{f7\}\}$. For tp4, as $f5 \sqsubseteq f4$, $f5$ is safely removed at line 6, and $R(tp4) \rightarrow \{\{f4\}\}$. Second, Fedra localizes fragments on endpoints in $E(tp)$. For tp1, $E(tp1) \rightarrow \{\{C1\}, \{C2\}\}$. For tp4, $E(tp4) \rightarrow \{\{C1, C3\}\}$. Figure 6.3 shows the execution plans encoded in *R(tp)* and *E(tp)*. Triple patterns like tp1, with more than one relevant fragment, represent unions in the execution plan.

Algorithm 17 Union reduction algorithm

```
Require: tps : set of TriplePattern; E : mapping from TriplePattern to set of set of Endpoint
19: procedure unionReduction(tps, E)
20:     triplesWithMultipleFragments ← { tp : tp ∈ tps ∧ cardinality(E(tp)) > 1 }
21:     for each tp ∈ triplesWithMultipleFragments do
22:         commonSources ← (⋂ f : f ∈ E(tp))          ▷ get sources in all subsets in E(tp)
23:         if commonSources ≠ ∅ then
24:             E(tp) ← { commonSources }
25:         end if
26:     end for
27: end procedure
```

Procedure unionReduction (cf. Algorithm 17) prunes non common endpoints, if possible, to access triple patterns from as few endpoints as possible. In our running example, it is not possible because there is no common endpoint that replicates both $f6$ and $f7$. However, if, for example, $f7$ were also replicated at $C1$, then only $C1$ would be selected to execute tp1.

Procedure bgpReduction (cf. Algorithm 18) transforms the join part of $E(tp)$ (cf. Figure 6.3) into a set covering problem (cf. line 30). Each triple pattern is an element of the set to cover, e.g., tp2, tp3, tp4 correspond to s2, s3, s4 (cf. Figure 6.4a). And for each endpoint in $E(tp)$, we include the subset of triple patterns associated with that endpoint, e.g., for endpoint C1 we include the subset {s2,s4} as relevant fragments tp2 and tp4 are replicated by C1 (cf. Figure 6.4b). Line 31

2. As DBpedia is not included in the federation for processing Q, only fragments $f6$ and $f7$ are available to retrieve data for tp_1 and the engine will not produce all the answers that would be produced using DBpedia.

Algorithm 18 Basic graph pattern reduction algorithm

```
Require: tps : set of TriplePattern; E : mapping from TriplePattern to set of set of Endpoint
28: procedure BGPREDUCTION(tps, E)
29:     triplesWithOneFragment ← { tp : tp ∈ tps ∧ cardinality(E(tp)) = 1 }
30:     (S, C) ← minimal set covering instance using triplesWithOneFragment◁E
31:     C' ← MINIMALSETCOVERING(S, C)
32:     selected ← get endpoints encoded by C'
33:     for each tp ∈ triplesWithOneFragment do
34:         E(tp) ← E(tp) ⋂ selected
35:     end for
36: end procedure
```

relies on an existing heuristic [Joh73] to find the minimum set covering. In our example, it computes *C'={{s2,s3,s4}}*. Line 32 computes the selected endpoints, in our example, selected={ C3 }.

Finally, (Algorithm 16, line 15) chooses among endpoints that provide the same fragment and reduces data redundancy. For query Q, the whole algorithm returns D_2 of Table 6.1b.

Proposition 6.2.1. *Algorithm 16 has a time complexity of $O(n.m^2)$, with n the number of triple patterns in the query, m the number of fragments, k the number of endpoints, l the number of basic graph patterns in the query, and $m \gg k \wedge k \gg l$ holds.*

The upper bound given in Proposition 6.2.1 is unlikely to be reached, as it requires for all fragments to be relevant for each of the triple patterns. In practice (e.g., experiments from Section 6.3), even for high number of fragments (> 450), the source selection time remains low (< 2 secs).

Theorem 1. *If all the RDF data accessible through the endpoints of a federation are described as replicated fragments, Fedra source selection leads query engine to produce complete answers wrt the federation data.*

6.3 Experimental Study

The goal of the experimental study is to evaluate the effectiveness of Fedra. We compare the performance of federated SPARQL queries using FedX, DAW [Sal+13] +FedX, Fedra +FedX, ANAPSID, DAW+ANAPSID, and Fedra +ANAPSID. DAW is a source selection ables to detect overlapping between datasets and optimize source selection based on that. However, DAW is not designed to manage data replication, there is no support for explicitly define and use replicated fragments. Therefore, DAW may select redundant data sources and generate a high number of intermediate results.

We expect to see that Fedra selects less sources than the engines and DAW, and transfers less data from endpoints to the query engines.

Table 6.2 – Dataset characteristics: version, number of different triples (# DT), and predicates (# P)

Dataset	Version date	# DT	# P
Diseasome	19/10/2012	72,445	19
Semantic Web Dog Food	08/11/2012	198,797	147
DBpedia Geo-coordinates	06/2012	1,900,004	4
LinkedMDB	18/05/2010	3,579,610	148
WatDiv1	_	104,532	86
WatDiv100	_	10,934,518	86

Datasets and Queries: We use the real datasets: Diseasome, Semantic Web Dog Food, LinkedMDB, and DBpedia Geo-coordinates. Further, we consider two instances of the Waterloo SPARQL Diversity Test Suite (WatDiv) synthetic dataset [Alu+14; Alu+13] with 10^5 and 10^7 triples. Table 6.2 shows the characteristics of these datasets. The datasets are hosted on local Linked Data Fragment (LDF) servers.

We generate 50,000 queries from 500 templates for the WatDiv federation. We remove the queries that caused engines to abort execution, and queries that returned zero results. For the real datasets, we generate more than 10,000 queries using PATH and STAR shaped templates with two to eight triple patterns, that are instantiated with random values from the datasets. We include the DISTINCT modifier in all the queries, in order to make them susceptible to a reduction in the set of selected sources without changing the query answer.

For each dataset, we setup a federation of ten consumer SPARQL endpoints (ten as in [Sal+13]). Consumer SPARQL endpoints are implemented using Jena Fuseki 1.1.1 [3]. Each consumer endpoint selects 100 random queries. Each triple pattern of the query is executed as a SPARQL construct query with the LDF client [4]. The results are stored locally if not present in at least three consumer endpoints and a fragment definition is created. This replication factor of three was set to avoid federations where all the fragments were replicated by all the endpoints.

In order to measure the number of transferred data, the federated query engine accesses data consumer endpoints through a proxy.

Implementations: FedX 3.0 [5] and ANAPSID [6] have been modified to call Fedra and DAW [Sal+13] source selection strategies during query processing. Thus, each engine can use the selected sources to perform its own optimization strategies. Fedra and DAW [7] are implemented in both Java 1.7

3. http://jena.apache.org/, January 2015.
4. https://github.com/LinkedDataFragments, March 2015.
5. http://www.fluidops.com/fedx/, September 2014.
6. https://github.com/anapsid/anapsid, September 2014.
7. We had to implement DAW as its code is not available.

and Python 2.7.3. Thus, Fedra and DAW are integrated in FedX (Java) and ANAPSID (Python), reducing the performance impact of including these new source selection strategies. Proxies are implemented in Java 1.7. using the Apache HttpComponents Client library 4.3.5 [8]. We used R [9] to compute the Wilcoxon signed rank test [Wil92].

Evaluation Metrics: *i*) *Number of Selected Sources (NSS):* is the sum of the number of sources that have been selected per triple pattern. *ii*) *Number of Transferred Tuples (NTT):* is the number of tuples transferred from all the endpoints to the query engine during a query execution.

Further informations (implementation, results, setups details, tests p-values) are available at https://sites.google.com/site/fedrasourceselection.

Data Redundancy Minimization

To measure the reduction of the number of selected sources, 100 queries were randomly chosen, and the source selection was performed for these queries for each federation using ANAPSID and FedX with and without Fedra or DAW. For each query, the sum of the number of selected sources per triple pattern was computed. Boxplots are used to present the results (Figure 6.5). Both Fedra and DAW significantly reduce the number of selected sources, however, the reduction achieved by Fedra is greater than the achieved by DAW.

To confirm it, we formulated the null hypothesis: "Fedra selects the same number of sources as DAW does", and performed a Wilcoxon signed rank test, p-values were inferior or equal to 1.4e-05 for all federations and engines. These low p-values allow for rejecting the null hypothesis that DAW and Fedra reduction are similar, and accepting the alternative hypothesis that Fedra reduction is greater than the one achieved by DAW. Fedra source selection strategy identifies the relevant fragments and endpoints that provide the same data. Only one of them is actually selected; in consequence, a huge reduction on the number of selected sources of up to 400% per query is achieved.

Data Transfer Minimization

To measure the reduction in the number of transferred tuples, queries were executed using proxies that measure the number of transmitted tuples from endpoints to the engines. Because queries that timed out have no significance on number of transferred tuples, we removed all these queries from

8. https://hc.apache.org/, October 2014.
9. http://www.r-project.org/

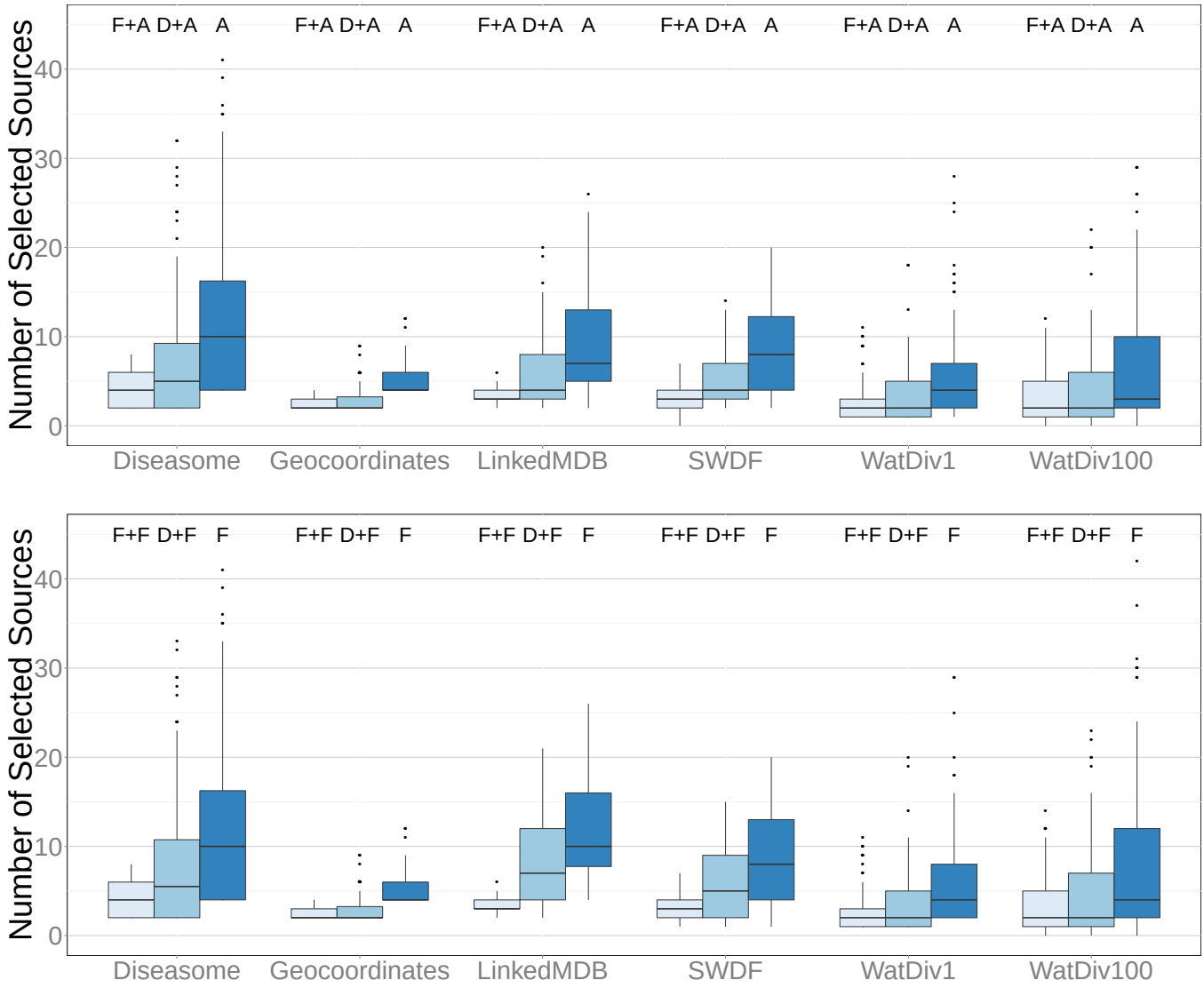


Figure 6.5 – Number of Selected Sources for execution of ANAPSID (A) and FedX (F) using Fedra (F+), DAW (D+), and the engine source selection

the study.[10] Results (Figure 6.6) show that Fedra source selection strategy leads to executions with considerably less intermediate results in all the federations except in the SWDF federation. In some queries of the SWDF federation, Lilac +FedX sends exclusive groups that include BGPs with triple patterns that do not share a variable, i.e., BGPs with Cartesian products; in presence of Cartesian product, large intermediate results may be generated. Queries with Cartesian products counters Fedra positive impact over other queries.

Despite that, globally Fedra shows an effective reduction of the number of transferred tuples. To confirm it, we formulated the null hypothesis: "using sources selected by Fedra leads to transfer the same number of tuples as using sources selected by DAW"; and performed a Wilcoxon signed rank test, p-values were inferior or equal to 0.002 for all federations and engines except SWDF

10. Up to six queries out of 100 queries did not successfully finish in 1,800 seconds, details available at the web page.

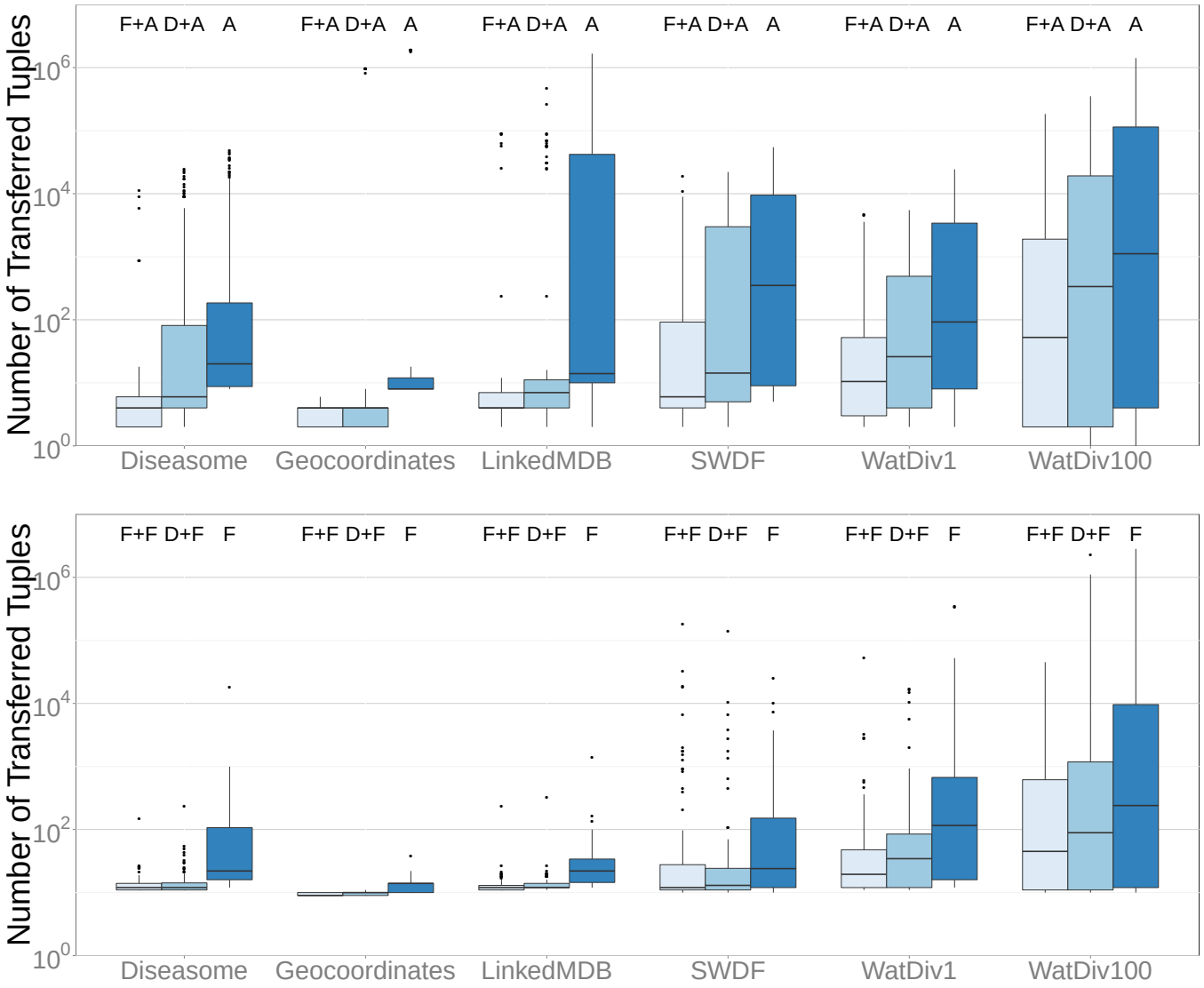


Figure 6.6 – Number of Transferred Tuples during execution with ANAPSID (A) and FedX (F) using Fedra (F+), DAW (D+), and the engine source selection

federation + FedX engine. In consequence, for all combinations of federation and engines except SWDF+FedX, we can reject the null hypothesis DAW and Fedra number of transferred tuples are similar and accept the alternative hypothesis that Fedra achieves a greater reduction of the number of transferred tuples than DAW. The reduction of the number of transferred tuples is mainly due to Fedra source selection strategy aims to find opportunities to execute joins in the endpoints, and mostly, it leads to a significant reduction of the intermediate results size of up to four orders of magnitude.

6.4 Conclusion

I illustrated how replicating fragments allow for data re-organization from different data sources to better fit query needs of data consumers. Then, I proposed a replication-aware federated query

engine by extending state-of-art federated query engine ANAPSID and FedX with Fedra, a source selection strategy that approximates SSP-FR.

Fedra exploits fragment localities to reduce intermediate results. Experimental results demonstrate that Fedra achieves significant reduction of intermediate results while leading to produce complete answers.

This work opens several perspectives. First, we made the assumption that replicated fragments are perfectly synchronized and cannot be updated. We can leverage this assumption and manage the problem of federated query processing with divergence [Mon+14b] or use the work on col-graph to synchronize data.

Several variants of SSP-FR can also be developed. SSP-FR does not differentiate between endpoints and the cost of accessing endpoints is considered the same. Finally, SSP-FR and Fedra can be extended to solve the source selection problem where the number of public endpoint accesses is minimized [Mon+14b].

Supervised PhD Thesis associated with this chapter

[Mon16] Gabriela Montoya. « Answering SPARQL Queries using View ». PhD thesis. Université de Nantes, Mar. 2016.

Publications associated with this chapter

International peer-reviewed journals

[Mon+17] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « Decomposing Federated Queries in Presence of Replicated Fragments ». In: *Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier* 42 (2017), pp. 1–18.

International peer-reviewed conferences

[Mon+15] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « Federated SPARQL Queries Processing with Replicated Fragments ». In: *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference*. Bethlehem, United States, Oct. 2015, pp. 36–51.

Technical reports

[Mon+14b] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. *Fedra: Query Processing for SPARQL Federations with Divergence*. Tech. rep. May 2014. URL: http://hal.univ-nantes.fr/hal-01022740.

7

Research Directions

Traditionally, Linked Open Data (LOD) providers make their datasets accessible through SPARQL servers. Data consumers can query a single data source or a federation of data sources. However, existing SPARQL servers suffer from the problems of availability and scalability [Ara+13]. Therefore, existing SPARQL servers are unreliable for building real linked data applications. Recently, Triple Pattern Fragment (TPF) [Ver+14a] approach is proposed to tackle this issue by balancing the cost of query processing between data providers and data consumers. In TPF, data are hosted in TPF servers providing low-cost publication of data, at the same time, SPARQL query processing is moved to the TPF clients side, *i.e.*, data consumers. Following TPF approach, a SPARQL query engine runs within a web browser of a TPF clients, this enables to share query processing between a TPF server and a TPF client. This approach establishes a trade-off between data availability and performances leveraging the “pressure" on data providers. TPF improves availability of data at the price of decreasing the performance of query processing.

The main objective of my research project is to remove this trade-off without requiring more ressources from data providers. In order to ensure availability and performance, I propose an approach where data consumers not only participate in SPARQL query processing but they also share their ressources. The idea is to take advantage of clients resources to improve performance. I propose

to build a federation of data consumers where members of a federation share their storage resources and their processing capabilities. This requires is to build an efficient decentralized federated query engine running in the browsers of data consumers.

Linked Data in the Fog

An efficient decentralized federated query engine running in the browsers of data consumers will break the state of the art tradeoff between availability and performances by proposing a fog of browsers.

Fog of browsers produce new way for hosting data and processing SPARQL queries. The main scientific challenge are :

Customized overlay networks for a fog of browsers. A fog of browsers will connect thousands of browsers in unstructured overlay network. I will use known techniques in distributed systems [Ber+10] to build this overlay. I propose to build a *Random Peer Sampling (RPS)* overlay network that maintains the membership among connected browsers. On the top of this overlay, other overlay networks can be built according to the applications needs. For instance, I can build a *Clustering Overlay Network (CON)* that clusters nodes according to the similarity of their queries. I experiment this approach by building a collaborative behavioral cache Cyclades in the context of Web applications. Results [FSM16] show that behavior cache reduces by the 20% the load on the data provider server. The next step of this challenge will be to define new similarities and clustering metrics not only based on queries but also on the infrastructure itself. A possible clustering metric could be the latency of the network. Latency could impact negatively the performance.

Dynamic replication and consistency in a fog of browsers. To improve performance, a TPF client may decide to locally materialize frequently used data fragments. This kind of dynamic fragment replication poses a number of problems concerning replicated data consistency. Consistency has already been addressed in [Ibá+14], but without considering the data distribution and communication costs. The replication strategy (what, where, when) is conditioned by the overlay network (see above) for deciding between fragment replication/materialization and sub-query delegation. Replication strategies impact federated query engine performance.

Federated query engine for a fog of browsers. A browser executes an infinite stream of queries that arrive at anytime. A browser can execute a query by itself or delegate it to a neighbor. Due

to browsers limitations, a browser can execute only one query at a time. The challenge is to find with low overhead, free neighbors to execute in parallel the local workload. However, parallel query execution must balance the load not only on clients side but also on the server side. The parallelism must not exceed the limit of the server. In addition, query decomposition and optimisation must take in consideration replicated data.

The main goal is to build a performant and reliable federated SPARQL query engine by exploiting the local processing power of web browsers.

8

Bibliography

This bibliography is divided in two parts. The first part corresponds to the author of the document categorized publications and the second part contains articles by other cited authors.

8.1 Publications by Hala Skaf-Molli (Categorized)

International editorial activities

[Lan+10] Christoph Lange, Jochen Reutelshoefer, Sebastian Schaffert, and Hala Skaf-Molli, eds. *5th Semantic Wiki Workshop (SemWiki 2010) at the 7th European Semantic Web Conference (ESWC 2010), Hersonissos, Greece, June 1st,2010. Proceedings*. Vol. 632. CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[Lan+09] Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, eds. *4th Semantic Wiki Workshop (SemWiki 2009) at the 6th European Semantic Web Conference (ESWC 2009), Hersonissos, Greece, June 1st, 2009. Proceedings*. Vol. 464. CEUR Workshop Proceedings. CEUR-WS.org, 2009.

[Lan+08] Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, eds. *Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008) at the 5th European Seman-*

tic Web Conference (ESWC 2008), Tenerife, Spain, June 2nd, 2008. Vol. 360. CEUR Workshop Proceedings. CEUR-WS.org, 2008.

Book chapters

[BVS] Christian Bizer, Maria-Esther Vidal, and Hala Skaf-Molli. « Linked Open Data ». In: *Encyclopedia of Database Systems, to appear*. Ed. by Ling Liu and M. Tamer Özsu.

[Tor+16] Diego Torres, Hala Skaf-Molli, Pascal Molli, and Alicia Diaz. « Discovering Wikipedia Conventions Using DBpedia Properties ». In: Revised Selected, Invited Papers of Semantic Web Collaborative Spaces: SWCS 2013, and SWCS 2014, Springer International Publishing, 2016, pp. 115–144.

[Cor+14] Amélie Cordier, Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Fadi Badra, Julien Cojan, Emmanuelle Gaillard, Laura Infante-Blanco, Pascal Molli, Amedeo Napoli, and Hala Skaf-Molli. « Taaable: a Case-Based System for personalized Cooking ». In: *Successful Case-based Reasoning Applications-2*. Vol. 494. Studies in Computational Intelligence. Springer, Jan. 2014, pp. 121–162. ISBN: 978-3-642-38735-7. DOI: 10.1007/978-3-642-38736-4_7. URL: http://hal.inria.fr/hal-00912767.

[Dan+04] Farhad Daneshgar, Pradeep Ray, Fethi Rahbi, Hala Skaf-Molli, Pascal Molli, and Claude Godart. « Knowledge Sharing Infrastructures for Teams within Virtual Communities ». In: *e-Collaborations and Virtual Organizations*. IGP/Infosci/IRM Press, 2004. URL: http://hal.inria.fr/inria-00108109.

[God+99] Claude Godart, Noureddine Belkhatir, Antonio Carzaniga, Jacky Estublier, Elisabetta Di Nitto, Jens H. Jahnke, Patricia Lago, Wilhelm Schäfer, and Hala Skaf. « Cooperation Control in PSEE ». In: *Software Process: Principles, Methodology, Technology*. Vol. 1500. Lecture Notes in Computer Science. Springer, 1999, pp. 117–164.

International peer-reviewed journals

[Mon+17] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « Decomposing Federated Queries in Presence of Replicated Fragments ». In: *Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier* 42 (2017), pp. 1–18.

[Dav+15] Alan Davoust, Hala Skaf-Molli, Pascal Molli, Babak Esfandiari, and Khaled Aslan. « Distributed Wikis: A Survey ». In: *Concurrency and Computation: Practice and Experience* 27 (2015), pp. 2751–2777. DOI: 10.1002/cpe.

[Mon+14a] Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « SemLAV: Local-as-View Mediation for SPARQL queries ». In: *LNCS Transactions on Large-Scale Data- and Knowledge-Centered Systems* 8420 (2014), pp. 33–58.

[Ibá+13] Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. « Live Linked Data: Synchronizing Semantic Stores with Commutative Replicated Data Types ». In: *International Journal of Metadata, Semantics and Ontologies* 8.2 (2013), pp. 119–133. URL: http://hal.inria.fr/hal-00903377.

[Naj+09] Hala Naja-Jazzar, Nishadi Desilva, Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. « OntoRest: A RST-based Ontology for Enhancing Documents Content Quality in Collaborative Writing ». In: *INFOCOMP Journal of Computer Science* 8.3 (2009), pp. 1–10.

[Mar+06a] Olivera Marjanovic, Hala Skaf-Molli, Pascal Molli, and Claude Godart. « Innovative Learning Designs Enabled by Process-Driven Collaborative Editing ». In: *Journal of Educational Technology and Society (endorsed by IEEE Learning Technology Task Force)* (2006).

[Baï+04] Karim Baïna, François Charoy, Claude Godart, Daniela Grigori, Saad El Hadri, Hala Skaf, S. Akifuji, Toshiaki Sakaguchi, Yoko Seki, and Masaichiro Yoshioka. « CORVETTE: a cooperative workflow for virtual teams coordination ». In: *International Journal of Networking and Virtual Organizations*. Special Issue on Infrastructures for New Virtual Organisations 2.3 (2004), pp. 232–245.

[God+04] Claude Godart, Pascal Molli, Gérald Oster, Olivier Perrin, Hala Skaf-Molli, Pradeep Ray, and Fethi Rabhi. « The ToxicFarm Integrated Cooperation Framework for Virtual Teams ». In: *Distributed and Parallel Databases* 15.1 (2004), pp. 67–88. DOI: 10.1023/B:DAPD.0000009432.79864.08. URL: http://hal.inria.fr/inria-00099944.

[SCG99] Hala Skaf, François Charoy, and Claude Godart. « Maintaining Shared Workspaces Consistency during Software Development ». In: *International Journal of Software Engineering and Knowledge Engineering*. Special Issue: Knowledge Discovery from Empirical Software Engineering Data 9.5 (1999), pp. 623–642. URL: https://hal.inria.fr/inria-00098802.

[Can+98] Gérôme Canals, Claude Godart, François Charoy, Pascal Molli, and Hala Skaf. « COO Approach to Support Cooperation in Software Developments ». In: *IEE Proceedings - Software* 145.2-3 (1998), pp. 79–84.

National peer-reviewed journals

[RSM09b] Charbel Rahhal, Hala Skaf-Molli, and Pascal Molli. « SWooki: Un Wiki Sémantique sur réseau Pair-à-Pair ». In: *Ingénierie des Systèmes d'Information* 14.1 (Feb. 2009), pp. 117–140.

International peer-reviewed conferences

[FSM16] Pauline Folz, Hala Skaf-Molli, and Pascal Molli. « CyCLaDEs: A Decentralized Cache for Triple Pattern Fragments ». In: *13th Extended Semantic Web Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2*. 2016, pp. 455–469.

[Mon+15] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « Federated SPARQL Queries Processing with Replicated Fragments ». In: *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference*. Bethlehem, United States, Oct. 2015, pp. 36–51.

[Ibá+14] Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. « Col-Graph: Towards Writable and Scalable Linked Open Data ». In: *ISWC - The 13th International Semantic Web Conference*. Riva del Garda, Italy, Oct. 2014, pp. 325–340.

[Le+13] Anh-Hoang Le, Marie Lefevre, Amélie Cordier, and Hala Skaf-Molli. « Collecting interaction traces in distributed semantic wikis ». In: *3rd International Conference on Web Intelligence, Mining and Semantics, WIMS '13, Madrid, Spain, June 12-14, 2013*. 2013, p. 21.

[Mon+13b] Gabriela Montoya, Luis-Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « GUN: An Efficient Execution Strategy for Querying the Web of Data ». In: *Database and Expert Systems Applications*. Springer. 2013, pp. 180–194.

[Tor+13] Diego Torres, Hala Skaf-Molli, Pascal Molli, and Alicia Diaz. « BlueFinder: Recommending Wikipedia Links Using DBpedia Properties ». In: *ACM Web Science Conference 2013 (WebSci 13)*. Paris, France, May 2013, pp. 115–144.

[ASM12] Khaled Aslan, Hala Skaf-Molli, and Pascal Molli. « Connecting Distributed Version Control Systems Communities to Linked Open Data ». In: *CTS 2012 - The International Conference on Collaboration Technologies and Systems - 2012*. 2012.

[Tor+12a] Diego Torres, P. Molli, H. Skaf-Molli, and A. Diaz. « From DBpedia to Wikipedia: Filling the Gap by Discovering Wikipedia Conventions ». In: *2012 IEEE/WIC/ACM International Conference on Web Intelligence (WI 12)*. 2012.

[Asl+11a] Khaled Aslan, Nagham Alhadad, Hala Skaf-Molli, and Pascal Molli. « SCHO: An Ontology Based Model for Computing Divergence Awareness in Distributed Collaborative Systems ». In: *European Conference on Computer-Supported Cooperative Work*. Aarhus, Danemark, Sept. 2011.

[Tor+11] Diego Torres, Alicia Diaz, Hala Skaf-Molli, and Pascal Molli. « Semdrops: A Social Semantic Tagging Approach for Emerging Semantic Data ». In: *2011 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2011)*. Lyon, France, Aug. 2011.

[ASM10] Khaled Aslan, Hala Skaf-Molli, and Pascal Molli. « From Causal History to Social Network in Distributed Social Semantic Software ». In: *Web Science Conference 2010 - WebSci10*. Apr. 2010.

[BBS10] Anne Boyer, Armelle Brun, and Hala Skaf-Molli. « Human Computer Collaboration to Improve Annotations in Semantic Wikis ». In: *6th Conference on Web Information Systems and Technologies (Webist 2010)*. Valencia, Spain, Apr. 2010, p. 8. URL: https://hal.inria.fr/inria-00378416.

[BSB10] Armelle Brun, Hala Skaf-Molli, and Anne Boyer. « Raising up Annotations In Pedagogical Resources by Human-Computer Collaboration ». In: *European Distance and*

E-learning Network (EDEN 2010). Budapest, Hungary, Oct. 2010. URL: https://hal.inria.fr/inria-00597285.

[SCM10a] Hala Skaf-Molli, Gérôme Canals, and Pascal Molli. « DSMW: a distributed infrastructure for the cooperative edition of semantic wiki documents ». In: *ACM Symposium on Document Engineering (DocEng 2010) (Demo)*. Manchester, Royaume-Uni: ACM, 2010, pp. 185–186.

[SCM10b] Hala Skaf-Molli, Gérôme Canals, and Pascal Molli. « DSMW: Distributed Semantic MediaWiki ». In: *ESWC (2)*. Ed. by Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache. Vol. 6089. Lecture Notes in Computer Science. Heraklion, Crete, Greece: Springer, 2010, pp. 426–430. ISBN: 978-3-642-13488-3.

[SCM10c] Hala Skaf-Molli, Gérôme Canals, and Pascal Molli. « DSMW: Distributed Semantic MediaWiki ». In: *7th Extended Semantic Web Conference (ESCW 2010) (Demo)*. Vol. 6089. Lecture Notes in Computer Science. Heraklion, Gréce: Springer, 2010.

[Rah+09a] Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, and Stephane Weiss. « Multi-Synchronous Collaborative Semantic Wikis ». In: *10th International Conference on Web Information Systems Engineering (WISE 2009)*. Vol. 5802. Lecture Notes in Computer Science. Poznan, Pologne: Springer, Oct. 2009, pp. 115–129.

[SRM09] Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. « Peer-to-peer Semantic Wikis ». In: *20th International Conference on Database and Expert Systems Applications - DEXA 2009*. Vol. 5690. Lecture Notes in Computer Science. Linz, Autriche: Springer-Verlag, Aug. 2009, pp. 196–213.

[Tor+09b] Diego Torres, Hala Skaf-Molli, Alicia Diaz, and Pascal Molli. « Supporting Personal Semantic Annotations in P2P Semantic Wikis ». In: *20th International Conference on Database and Expert Systems Applications - DEXA 2009*. Vol. 5690. Lecture Notes in Computer Science. Linz, Autriche: Springer Berlin / Heidelberg, Aug. 2009, pp. 317–331.

[Ska+08] Hala Skaf-Molli, Pascal Molli, Charbel Rahhal, and Hala Naja-Jazzar. « Collaborative Writing of XML Documents ». In: *IEEE 3rd International Conference on Information*

& Communication Technologies : From Theory to Applications - ICTTA 2008. Damas, Syrie: IEEE, Apr. 2008, pp. 1–6. DOI: 10.1109/ICTTA.2008.4530334.

[Ost+07] Gérald Oster, Hala Skaf-Molli, Pascal Molli, and Hala Naja-Jazzar. « Supporting Collaborative Writing of XML Documents ». In: *9th International Conference on Enterprise Information Systems - ICEIS 2007*. Funchal, Madeira, Portugal, June 2007, pp. 335–341.

[Rah+07] Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, and Nishadi Desilva. « SemCW: Semantic Collaborative Writing using RST ». In: *The 3rd International Conference on Collaborative Computing:Networking, Applications and Worksharing - CollaborateCom'2007*. IEEE, Nov. 2007, pp. 484–493.

[Ska+07] Hala Skaf-Molli, Claudia Ignat, Charbel Rahhal, and Pascal Molli. « New Work Modes For Collaborative Writing ». In: *International Conference on Enterprise Information Systems and Web Technologies - EISWT-07*. Orlando, USA: ISRST, July 2007, pp. 176–182.

[Mar+06b] Olivera Marjanovic, Hala Skaf-Molli, Pascal Molli, Fethi Rabhi, and Claude Godart. « Supporting Complex Collaborative Learning Activities: The LIBRESOURCE Approach ». In: *8th International Conference on Enterprise Information Systems, ICEIS 2006*. Paphos- Cyprus, May 2006.

[Ska+06] Hala Skaf-Molli, Pascal Molli, Olivera Marjanovic, and Claude Godart. « LibreSource: Web Based platform for Supporting Collaborative Activities ». In: *2nd IEEE International Conference on Information and Communication Technologies: From Theory to Applications - ICTTA 2006*. Vol. 2. Damas - Syrie: IEEE, Apr. 2006, pp. 3309–3313. URL: http://hal.inria.fr/inria-00097435.

[Mol+03] Pascal Molli, Gérald Oster, Hala Skaf-Molli, and Abdessamad Imine. « Using the Transformational Approach to Build a Safe and Generic Data Synchronizer ». In: *International Conference on Supporting Group Work - Group 2003*. Sanibel Island, Florida, USA: ACM Press, Nov. 2003, pp. 212–220. DOI: 10.1145/958160.958194.

[Ska+03] Hala Skaf-Molli, Pascal Molli, Gérald Oster, Claude Godart, Pradeep Ray, and Fethi Rabhi. « Toxic Farm : a cooperative management platform for virtual teams and

enterprises ». In: *5th International Conference on Enterprise Information Systems - ICEIS'03*. Angers, France: none, 2003. URL: http://hal.inria.fr/inria-00099802.

[MSO02] Pascal Molli, Hala Skaf-Molli, and Gérald Oster. « Divergence Awareness for Virtual Team through the Web ». In: *Sixth World Conference on Integrated Design and Process Technology - IDPT'2002*. Pasadena, CA, USA: Society for Design & Process Science (SDPS), June 2002, 10 p. URL: http://hal.inria.fr/inria-00100747.

[Mol+02] Pascal Molli, Hala Skaf-Molli, Gérald Oster, and Sébastien Jourdain. « SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments ». In: *Seventh International Conference on Computer Supported Cooperative Work in Design - CSCWD'02*. Rio de Janeiro, Brasil: none, 2002, 5 p. URL: http://hal.inria.fr/inria-00107571.

[Mol+01] Pascal Molli, Hala Skaf-Molli, Claude Godart, Pradeep Ray, Rajan Shankaran, and Vijay Varadharajan. « Integrating Network Services for Virtual Teams ». In: *International Conference on Enterprise Information Systems - ICEIS 2001*. Setúbal, Portugal, 2001, 6 p. URL: http://hal.inria.fr/inria-00147539.

[God+00] Claude Godart, François Charoy, Olivier Perrin, and Hala Skaf. « Cooperative Workflows to Coordinate Asynchronous Cooperative Applications in a Simple Way ». In: *Seventh International Conference on Parallel & Distributed Systems - ICPADS 2000*. Parallel and Distributed Systems, 2000. Proceedings. Seventh International Conference on. IEEE Computer Society. Iwate, Japan: IEEE, July 2000, pp. 409–416.

[GSC00] Daniela Grigori, Hala Skaf-Molli, and François Charoy. « Adding Flexibility in a Cooperative Workflow Execution Engine ». In: *8th International Conference High-Performance Computing and Networking HPCN Europe*. Amsterdam, The Netherlands, May 2000, pp. 227–236.

[SCG97] Hala Skaf, François Charoy, and Claude Godart. « An Hybrid Approach to Maintain Consistency of Cooperative Software Development Activities ». In: *Ninth International Conference on Software Engineering and Knowledge Engineering - SEKE97*. Madrid, Spain, June 1997.

[God+96] Claude Godart, Gérôme Canals, François Charoy, Pascal Molli, and Hala Skaf. « Designing and Implementing *COO*: Design Process, Architectural Style, Lessons Learned ». In: *18th International Conference on Software Engineering, Berlin, Germany, March 25-29, 1996, Proceedings*. 1996, pp. 342–352. URL: http://portal.acm.org/citation.cfm?id=227726.227796.

[SCG96] Hala Skaf, Francois Charoy, and Claude Godart. « Maintaining Consistency of Cooperative Software Development Activities. » In: *6th International Workshop on Foundations of Models and Languages for Data and Objects, Schloss Dagstuhl, Germany, September 16-20*. 1996, pp. 103–118.

[Can+95] Gérôme Canals, François Charoy, Claude Godart, Frank Juillard, Pascal Molli, Alexandre Rossel, and Hala Skaf. « P-Root & COO: un système de gestion d'objets et des services de mise en oeuvre de procédé de développement ». In: *Onzièmes Journées Bases de Données Avancées, 29 Août - 1er Septembre 1995, Nancy (Informal Proceedings)*. 1995, p. 477.

National peer-reviewed conferences

[Mon+16] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « Federated SPARQL Queries Processing with Replicated Fragments ». In: *32ème Conférence sur la Gestion de Données - Principes, Technologies et Applications (BDA)*. Potières, France, Oct. 2016, pp. 36–51.

[Fol+14a] Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, and Maria-Esther Molli Pascal and. « SemLAV : Interroger le Web profond et le Web des données avec SPARQL ». In: *Journées Francophones BDA : Base de Données Avancées*. Grenoble, France, Oct. 2014. URL: https://hal.archives-ouvertes.fr/hal-01089917.

[SMC09] Hala Skaf-Molli, Pascal Molli, and Gérôme Canals. « SWooki: Supporting Disconnection in a Peer-to-peer Semantic Wiki ». In: *5èmes Journées Francophones Mobilité et Ubiquité 2009 - UbiMob'09*. Lille, France, July 2009.

[Jou+06] Florent Jouille, Bernard Ganne, Hala Skaf-Molli, and Pascal Molli. « LibreSource: Une plate-forme de développement collaboratif basée sur le WEB ». In: *19èmes Journées In-*

ternationales "Génie Logiciel & Ingénierie de Systèmes et leurs Applications" - ICSSEA 2006. Paris/France, Dec. 2006.

[Ost+04] Gérald Oster, Pascal Molli, Hala Skaf-Molli, and Abdessamad Imine. « Un modèle sûr et générique pour la synchronisation de données divergentes ». In: *Premières Journées Francophones : Mobilité et Ubiquité - UbiMob'04*. Nice, France, June 2004, 9 p.

[Boi+94] Olivier Boissier, Yves Demazeau, Gérald Masini, and Hala Skaf. « Une architecture Multi-Agents pour l'implémentation du bas niveau d'un système de compréhension de scènes ». In: *Deuxièmes Journées Francophones Intelligence Artificielle Distribuée et Systèmes Multi-Agents (JFIADSMA'94)*. Voiron, France, May 1994, pp. 293–304.

[Mas+93] G. Masini, H. Skaf, Y. Demazeau, and O. Boissier. « Contribution à l'implantation d'un système de compréhension de scènes par une architecture multi-agents ». In: *Journées ORASIS PRC-CHM Pôle Vision*. Mulhouse, Oct. 1993.

International peer-reviewed workshops

[Fol+15] Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « Parallel Data Loading during Querying Deep Web and Linked Open Data with SPARQL ». In: *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, PA, USA, October 11, 2015*. 2015, pp. 63–74.

[Fol+14b] Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. « SemLAV: Querying Deep Web and Linked Open Data with SPARQL ». In: *ESWC: Extended Semantic Web Conference, ESWC 2014 Satellite Events, LNSC 476*. May 2014, pp. 332–337.

[Cha+12] Pierre-Antoine Champin, Amélie Cordier, Elise Lavoué, Marie Lefevre, and Hala Skaf-Molli. « User assistance for collaborative knowledge construction ». In: *Workshop on Semantic Web Collaborative Spaces (SWCS), in conjunction with the World Wide Web 2012 International Conference*. Lyon, France: ACM, Apr. 2012, pp. 1065–1074. URL: https://hal.archives-ouvertes.fr/hal-00692091.

[Dan+12] Ibáñez Luis Daniel, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. « Synchronizing Semantic Stores with Commutative Replicated Data Types ». In: *SWCS - Semantic Web Collaborative Spaces Workshop - 2012*. Lyon, France: ACM, 2012, pp. 1091–1096. URL: http://hal.inria.fr/hal-00686484.

[Ska+12] Hala Skaf-Molli, Emmanuel Desmontils, Emmanuel Nauer, Gérôme Canals, Amélie Cordier, Marie Lefevre, Pascal Molli, and Yannick Toussaint. « Knowledge Continuous Integration Process (K-CIP) ». In: *WWW 2012 - SWCS'12 Workshop - 21st World Wide Web Conference - Semantic Web Collaborative Spaces workshop*. Lyon, France, Apr. 2012, pp. 1075–1082. URL: http://hal.inria.fr/hal-00765596.

[Tor+12b] Diego Torres, Pascal Molli, Hala Skaf-Molli, and Alicia Diaz. « Improving Wikipedia with DBpedia ». In: *SWCS - Semantic Web Collaborative Spaces Workshop 2012 in 21st WWW Conference 2012*. Lyon, France, Apr. 2012.

[Asl+11b] Khaled Aslan, Pascal Molli, Hala Skaf-Molli, and Stéphane Weiss. « C-Set : a Commutative Replicated Data Type for Semantic Stores ». In: *RED: Fourth International Workshop on REsource Discovery at 8th Extended Semantic Web Conference, ESWC 2011*. Heraklion, Grèce, May 2011.

[Bla+10a] Alexandre Blansche, Julien Cojan, Valmi Dufour-Lussier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli, and Yannick Toussaint. « TAAABLE 3: Adaptation of ingredient quantities and of textual preparations ». In: *18h International Conference on Case-Based Reasoning - ICCBR 2010, Computer Cooking Contest Workshop Proceedings*. Alessandria, Italie, 2010. URL: http://hal.inria.fr/inria-00526663.

[Bla+10b] Alexandre Blansche, Hala Skaf-Molli, Pascal Molli, and Amedeo Napoli. « Human-machine Collaboration for Enriching Semantic Wikis using Formal Concept Analysis ». In: *5th Workshop on Semantic Wikis Linking Data and People - SemWiki2010*. 2010.

[Bad+09] Fadi Badra, Julien Cojan, Amélie Cordier, Jean Lieber, Thomas Meilender, Alain Mille, Pascal Molli, Emmanuel Nauer, Amedeo Napoli, Hala Skaf-Molli, and Yannick Toussaint. « Knowledge acquisition and discovery for the textual case-based cooking system

WIKITAAABLE ». In: *8th International Conference on Case-Based Reasoning - ICCBR 2009, Workshop Proceedings*. Seattle, USA, July 2009, pp. 249–258.

[Cor+09a] Amélie Cordier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli, and Yannick Toussaint. « WIKITAAABLE: A semantic wiki as a blackboard for a textual case-based reasoning system ». In: *SemWiki 2009 - 4rd Semantic Wiki Workshop at the 6th European Semantic Web Conference - ESWC 2009*. Heraklion, Grèce, May 2009.

[Rah+09b] Charbel Rahhal, Stéphane Weiss, Hala Skaf-Molli, Pascal Urso, and Pascal Molli. « Undo in Peer-to-peer Semantic Wikis ». In: *SemWiki' 2009 - 4rd Semantic Wiki Workshop at the 6th European Semantic Web Conference - ESWC 2009*. Heraklion, Grèce, June 2009.

[Tor+09a] Diego Torres, Alicia Diaz, Hala Skaf-Molli, and Pascal Molli. « Personal Navigation in Semantic Wikis ». In: *International Workshop on Adaptation and Personalization for Web 2.0 - AP-WEB 2.0 2009*. Vol. 485. CEUR Workshop Proceedings. Trento, Italie: CEUR-WS.org, June 2009, pp. 148–151.

[RSM08a] Charbel Rahhal, Hala Skaf-Molli, and Pascal Molli. « SWOOKI: A Peer-to-peer Semantic Wiki ». In: *3rd Semantic Wiki Workshop (SemWiki'2008) at the 5th European Semantic Web Conference (ESWC 2008)*. Ed. by Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel. Vol. 360. CEUR Workshop Proceedings. Tenerife, Espagne: CEUR-WS.org, June 2008, pp. 124–126.

[RSM08b] Charbel Rahhal, Hala Skaf-Molli, and Pascal Molli. « SWOOKI: A Peer-to-peer Semantic Wiki ». In: *The 3rd Semantic Wikis workshop, co-located with the 5th Annual European Semantic Web Conference (ESWC), Tenerife, Spain*. 2008.

[Ign+07] Lavinia Ignat Claudia, Gérald Oster, Pascal Molli, and Hala Skaf-Molli. « A Collaborative Writing Mode for Avoiding Blind Modifications ». In: *9th International Workshop on Collaborative Editing Systems - IWCES 2007*. Sanibel Island, Florida, USA, 2007.

[DS06] Nishadi De-Silva and Hala Skaf-Molli. « Narratives to preserve coherence in collaborative writing ». In: *The Eighth International Workshop on Collaborative Editing Systems - ACM CSCW 2006*. Banff, Canada, Nov. 2006.

[SMO03] Hala Skaf-Molli, Pascal Molli, and Gérald Oster. « Semantic Consistency for Collaborative Systems ». In: *Fifth International Workshop on Collaborative Editing - ECSCW'2003*. Helsinki, Finland: none, 2003, 8 p.

[MSB01] Pascal Molli, Hala Skaf-Molli, and Christophe Boutier. « State Treemap: an Awareness Widget for Multi-Synchronous Groupware ». In: *7th International Workshop on Groupware - CRIWG'2001*. Darmstadt, Germany: IEEE, Sept. 2001, pp. 106–114. DOI: 10.1109/CRIWG.2001.951823.

[BSM99] Abdelmajid Bouazza, Hala Skaf-Molli, and Pascal Molli. « Coordinating Virtual Teams by Measuring Group Divergence ». In: *Workshop on Groupware related Task Design at GROUP'99 Conference*. Phoenix, Arizona, USA: none, 1999, 4 p. URL: http://hal.inria.fr/inria-00098869.

[SCG98] Hala Skaf, François Charoy, and Claude Godart. « Flexible Integrity Control of Cooperative Applications ». In: *The Ninth International Workshop on Database & Expert Systems Applications*. 9th International Workshop on Database and Expert Systems Applications (DEXA'98). Vienne, Autriche: IEEE, Aug. 1998, pp. 901–906.

National peer-reviewed workshops

[Cor+09b] Amélie Cordier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli, and Yannick Toussaint. « WikiTaaable, un wiki sémantique utilisé comme un tableau noir dans un système de raisonnement à partir de cas textuel ». In: *17ème atelier de Raisonnement à Partir de Cas - RàPC 2009*. Ed. by Béatrice Fuchs and Amedeo Napoli. Paris, France, June 2009.

[SRM08] Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. « Wiki sémantique sur un réseau pair-à-pair ». In: *Atelier IC2.0 en association avec le 19èmes journées Francophone d'Ingénierie des Connaissances - IC 2008*. Nancy, France, June 2008, 4p.

Technical reports and others

[Mon+14b] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. *Fedra: Query Processing for SPARQL Federations with Divergence*. Tech. rep. May 2014. URL: http://hal.univ-nantes.fr/hal-01022740.

[ASM13] Khaled Aslan-Almoubayed, Hala Skaf-Molli, and Pascal Molli. *GroupDiv: Formalizing and Computing Group Divergence Awareness in Multi-Synchronous Distributed Collaborative Systems*. Tech. rep. July 2013. URL: http://hal.univ-nantes.fr/hal-00842714.

[Mon+13a] Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. *SemLAV: Local-as-View Mediation for SPARQL queries*. Tech. rep. July 2013. URL: http://hal.univ-nantes.fr/hal-00841985.

[SM12] Hala Skaf-Molli and Pascal Molli. *Distributed Semantic Wiki: Kolflow Project -Task 5- State of the art (D5.1)*. Tech. rep. June 2012. URL: http://hal.archives-ouvertes.fr/hal-00707185.

[RSM09a] Charbel Rahhal, Hala Skaf-Molli, and Pascal Molli. *Multi-synchronous Collaborative Semantic Wikis*. Tech. rep. RR-6947. INRIA, 2009, p. 22.

8.2 Publications by other authors

[Leh+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. « DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia ». In: *Semantic Web Journal* 6.2 (2015), pp. 167–195.

[Alu+14] Günes Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. « Diversified Stress Testing of RDF Data Management Systems ». In: *ISWC 2014, Part I*. 2014, pp. 197–212.

[Ver+14a] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. « Querying Datasets on the Web with High Availability ». In: *ISWC 2014, Part I*. 2014, pp. 180–196.

[Ver+14b] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. « Web-Scale Querying through Linked Data Fragments ». In: *Linked Data on the Web Workshop (LDOW)*. 2014.

[WCG14] Marcin Wylot, Philippe Cudré-Mauroux, and Paul Groth. « TripleProv: Efficient Processing of Lineage Queries in a Native RDF Store ». In: *WWW*. 2014.

[Aco+13] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. « Crowdsourcing Linked Data Quality Assessment ». In: *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*. 2013, pp. 260–276.

[Alu+13] Günes Aluç, M Tamer Ozsu, Khuzaima Daudjee, and Olaf Hartig. « chameleon-db: a Workload-Aware Robust RDF Data Management System ». In: *University of Waterloo, Tech. Rep. CS-2013-10* (2013).

[Ara+13] Carlos Buil Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. « SPARQL Web-Querying Infrastructure: Ready for Action? » In: *International Semantic Web Conference ISWC2013*. 2013, pp. 277–293. DOI: 10.1007/978-3-642-41338-4_18.

[BO13] Tim Berners-Lee and Kieron O'Hara. « The read-write Linked Data Web ». In: *Philosophical Transactions of the Royal Society* (2013).

[Kar+13] Grigoris Karvounarakis, Todd J. Green, Zachary G. Ives, and Val Tannen. « Collaborative Data Sharing via Update Exchange and Provenance ». In: *ACM Transactions on Database Systems* 38.3 (2013).

[Sal+13] Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, Josiane Xavier Parreira, Helena F. Deus, and Manfred Hauswirth. « DAW: Duplicate-AWare Federated Query Processing over the Web of Data ». In: *ISWC 2013, Part I*. 2013, pp. 574–590.

[Lu+12] W. Lu, Y. Shen, S. Chen, and B.C. Ooi. « Efficient processing of k nearest neighbor joins using MapReduce ». In: *Proceedings of the VLDB Endowment* 5.10 (2012), pp. 1016–1027.

[Aco+11] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. « ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints ». In: *International Semantic Web Conference (1)*. 2011, pp. 18–34.

[GIT11] Todd J. Green, Zachary G. Ives, and Val Tannen. « Reconcilable Differences ». In: *Theory of Computer Systems* 49.2 (2011).

[KA11] George Konstantinidis and José Luis Ambite. « Scalable query rewriting: a graph-based approach ». In: *SIGMOD Conference*. Ed. by Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis. ACM, 2011, pp. 97–108. ISBN: 978-1-4503-0661-4.

[Le+11] Wangchao Le, Songyun Duan, Anastasios Kementsietsidis, Feifei Li, and Min Wang. « Rewriting queries on SPARQL views ». In: *WWW*. Ed. by Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar. ACM, 2011, pp. 655–664. ISBN: 978-1-4503-0632-4.

[Sch+11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. « FedX: Optimization Techniques for Federated Query Processing on Linked Data ». In: *International Semantic Web Conference (1)*. 2011, pp. 601–616.

[Sha+11] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. « Conflict-Free Replicated Data Types ». In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 2011, pp. 386–400.

[Ber+10] Marin Bertier, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. « The Gossple Anonymous Social Network ». In: *11th International Middleware Conference 'Middleware 2010*. Vol. 6452. LNCS. Springer, 2010, pp. 191–211.

[IVB10] Daniel Izquierdo, Maria-Esther Vidal, and Blai Bonet. « An Expressive and Efficient Solution to the Service Selection Problem ». In: *International Semantic Web Conference (1)*. Ed. by Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm. Vol. 6496. Lecture Notes in Computer Science. Springer, 2010, pp. 386–401. ISBN: 978-3-642-17745-3.

[BHB09] Christian Bizer, Tom Heath, and Tim Berners-Lee. « Linked Data - The Story So Far ». In: *International Journal of Semantic Web Information Systems* 5.3 (2009), pp. 1–22.

[Biz+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. « DBpedia - A crystallization point for the Web of Data ». In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 154–165. ISSN: 1570-8268. DOI: 10.1016/j.websem.2009.07.

002. URL: http://www.sciencedirect.com/science/article/pii/S1570826809000225.

[BS09] Christian Bizer and Andreas Schultz. « The Berlin SPARQL Benchmark ». In: *Int. J. Semantic Web Inf. Syst.* 5.2 (2009), pp. 1–24.

[Bre+09] John G. Breslin, Uldis Bojārs, Alexandre Passant, Sergio Fernandez, and Stefan Decker. « SIOC: Content Exchange and Semantic Interoperability Between Social Networks ». In: *W3C Workshop on the Future of Social Networking*. Jan. 2009.

[BPD09] John G. Breslin, Alexandre Passant, and Stefan Decker. *The Social Semantic Web*. Springer Publishing Company, 2009.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. « Semantics and complexity of SPARQL ». In: *ACM Transaction On Database Systems (TODS)* 34.3 (2009), 16:1–16:45. ISSN: 0362-5915. DOI: 10.1145/1567274.1567278. URL: http://doi.acm.org/10.1145/1567274.1567278.

[WUM09] Stéphane Weiss, Pascal Urso, and Pascal Molli. « Logoot : a Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks ». In: *32nd International Conference on Distributed Computing Systems*. IEEE Computer Society, 2009, pp. 404–412.

[Buf+08] Michel Buffa, Fabien L. Gandon, Guillaume Ereteo, Peter Sander, and Catherine Faron. « SweetWiki: A semantic wiki ». In: *Journal of Web Semantics* 6.1 (2008), pp. 84–97.

[P+08] Eric Prud'Hommeaux, Andy Seaborne, et al. « SPARQL query language for RDF ». In: *W3C recommendation* 15 (2008).

[GKT07] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. « Provenance Semirings ». In: *Principles of Database Systems (PODS)*. 2007.

[He+07] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. « Accessing the Deep Web ». In: *ACM Communication* 50.5 (2007), pp. 94–101.

[Krö+07] Markus Krötzsch, Denny Vrandecic, Max Völkel, Heiko Haller, and Rudi Studer. « Semantic Wikipedia ». In: *Journal of Web Semantic* 5.4 (2007), pp. 251–261.

[WUM07] Stéphane Weiss, Pascal Urso, and Pascal Molli. « Wooki: a P2P Wiki-based Collaborative Writing Tool ». In: *Web Information Systems Engineering*. Nancy, France, 2007.

[ABV06] Yolifé Arvelo, Blai Bonet, and Maria-Esther Vidal. « Compilation of Query-Rewriting Problems into Tractable Fragments of Propositional Logic ». In: *AAAI*. AAAI Press, 2006, pp. 225–230.

[AT05] G. Adomavicius and A. Tuzhilin. « Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions ». In: *IEEE Transactions on Knowledge and Data Engineering* 17.6 (2005), pp. 734–749.

[LMS05] P. Leach, M. Mealling, and R. Salz. « A Universally Unique IDentifier (UUID) URN Namespace ». In: *Internet RFCs* RFC 4122 (2005). URL: http://www.rfc-editor.org/rfc/rfc4122.txt.

[SS05a] Yasushi Saito and Marc Shapiro. « Optimistic Replication ». In: *ACM Computing Surveys* 37.1 (2005), pp. 42–81. ISSN: 0360-0300.

[SS05b] Yasushi Saito and Marc Shapiro. « Optimistic replication ». In: *ACM Computer Survey* 37.1 (2005), pp. 42–81.

[MM04] Frank Manola and Eric Miller. « Resource description framework (RDF) primer ». In: *W3C Recommendation* 10 (2004).

[P+04] Peter F Patel-Schneider, Patrick Hayes, Ian Horrocks, et al. « OWL web ontology language semantics and abstract syntax ». In: *W3C recommendation* 10 (2004).

[B+01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. « The semantic web ». In: *Scientific american* 284.5 (2001), pp. 28–37.

[Hal01] Alon Y. Halevy. « Answering queries using views: A survey ». In: *VLDB J.* 10.4 (2001), pp. 270–294.

[PH01] Rachel Pottinger and Alon Y. Halevy. « MiniCon: A scalable algorithm for answering queries using views ». In: *VLDB J.* 10.2-3 (2001), pp. 182–198.

[OJ00] Malcolm Otter and Hilary Johnson. « Lost in hyperspace: metrics and mental models ». In: *Interacting with computers* 13.1 (2000), pp. 1–40.

[LC98] Kevin Larson and Mary Czerwinski. « Web page design: implications of memory, structure and scent for information retrieval ». In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '98. Los Angeles, California, USA:

ACM Press/Addison-Wesley Publishing Co., 1998, pp. 25–32. ISBN: 0-201-30987-4. DOI: 10.1145/274644.274649. URL: http://dx.doi.org/10.1145/274644.274649.

[Sun+98] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. « Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems ». In: *ACM Transactions on Computer-Human Interaction* 5.1 (1998), pp. 63–108. ISSN: 1073-0516. DOI: http://doi.acm.org/10.1145/274444.274447.

[GJM96] Ashish Gupta, H.V. Jagadish, and Inderpal Singh Mumick. « Data Integration using Self-Maintainable Views ». In: *International Conference on Extending Database Systems (EDBT)*. 1996.

[Dou95] Paul Dourish. « The Parting of the Ways: Divergence, Data Management and Collaborative Work ». In: *Proceedings of the European Conference on Computer-Supported Cooperative Work - ECSCW'95*. Stockholm, Sweden, 1995, pp. 215–230.

[Wie92] Gio Wiederhold. « Mediators in the Architecture of Future Information Systems ». In: *IEEE Computer* 25.3 (1992), pp. 38–49.

[Wil92] Frank Wilcoxon. « Individual comparisons by ranking methods ». In: *Breakthroughs in Statistics*. Springer, 1992, pp. 196–202.

[Mat89] Friedemann Mattern. « Virtual Time and Global States of Distributed Systems ». In: *Parallel and Distributed Algorithms* 1.23 (1989). Ed. by Michel Cosnard, Patrice Quinton, Yves Robert, and MichelEditors Raynal, pp. 215–226.

[EL88] Douglas Engelbart and Harvey Lehtman. « Working together ». In: *Byte* 13.13 (1988).

[LN85] Thomas K Landauer and DW Nachbar. « Selection from alphabetic and numeric menu trees using a touch screen: breadth, depth, and width ». In: *ACM SIGCHI Bulletin* 16.4 (1985), pp. 73–78.

[Joh73] David S. Johnson. « Approximation Algorithms for Combinatorial Problems ». In: *ACM Symposium on Theory of Computing*. Ed. by Alfred V. Aho et al. ACM, 1973, pp. 38–49.

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Hala SKAF-MOLLI

Gestion décentralisée de données du web sémantique

Decentralized Data Management for the Semantic Web

Résumé

Le web sémantique est une extension du web où l'information a une signification précise. Des milliers de jeux de données en RDF sont accessibles sur le web. Cependant, des problèmes important liés à la qualité des données, l'accès au web profond et la disponibilité des données restent ouverts. Pour la qualité de données, nous proposons de transformer le web de données vers un web de données en lecture/écriture. Un consommateur de données est à même de corriger une erreur. Nous définissions des algorithmes de synchronisation adaptés au modèle RDF. Pour l'accès au web profond, nous proposons une approche médiateur permettant de combiner données sémantique et données du web profond. Pour la disponibilité des données, nous proposons un modèle de réplication pour le web de données. Le problème est d'optimiser des requêtes fédérées en présence de réplicas.

Mots clés

Données ouvertes liées, RDF, Réplication, Requêtes SPARQL fédérées.

Abstract

The semantic web is an extension of the web where information has a precise meaning. Thousands of linked datasets are available on the web. Important problems concerning quality, deep web access and availability still unsolved. For data quality, we propose to transform the web of data into a read/write web of data. A data consumer will able to correct an error. Allowing consumers to write the semantic web poses the problem of data consistency. We define synchronization algorithms for RDF data model. To access to the deep web, we propose a mediator approach allowing to combine semantic data and deep web data. The problem is to improve the performance of queries in the presence of a large number of data sources. Finally, to ensure the availability, we propose a replication model for the web of data. The problem is to optimize federated SPARQL queries in the presence of replicas selected at queries execution time.

Key Words

Linked Open Data, RDF, Replication, Federated SPARQL queries.