A Second Generation of Peer-to-Peer Semantic Wikis

Charbel Rahhal(✉)

Faculty of Science, Lebanese University, Zahlé, Lebanon
charbelrahhal@gmail.com

Abstract. P2P Semantic Wikis (P2PSW) constitute a collaborative editing tool for knowledge and ontology creation, share and management. They ensure a massive collaboration in a distributed manner on replicated data composed of semantic wikis pages and semantic annotations. P2PSW are an instantiation of the optimistic replication model for semantic wikis. They ensure eventual syntactical consistency, i.e. that the wiki pages and semantic annotations store of the peers will eventually become identical. In spite of their advantages, these Wikis do not support a mechanism to maintain the quality of their semantic annotations. Thus, the content of the semantic wiki pages could be inconsistent for many reasons: the merge of the changes is made automatically by the wiki not by the users, missing information or inconsistent information added by the users of the peers. In this paper, I present a semantic inconsistency detection mechanism (SIDM) developed for P2PSW. SIDM detects the semantic inconsistency of the annotations in the semantic pages and improves the quality of the knowledge and the functionality of P2PSW. It indicates not only the existence of the semantic inconsistency in the wiki pages but also specifies the reason of the inconsistency. SIDM also facilitates the semantic inconsistency removal by determining exactly the position of the inconsistent annotations in the wiki pages and highlighting them via a semantic inconsistency visualization mechanism we developed.

Keywords: Semantic web · P2P semantic wikis · Semantic consistency

1 Introduction

P2P Semantic Wikis (P2PSW) [1] constitute a collaborative editing tool for knowledge and ontology creation, share and management. They ensure a massive collaboration in a distributed manner on a replicated data composed of semantic wikis pages and semantic annotations. In P2PSW, the number of peers can be very large, it can grow to thousands of thousands of peers. This happens without affecting the scalability and the functionality of the wiki. Research academics can work on common research projects and collaborate to produce their publications, people of same interests can produce and share same knowledge,

A. Hameurlain et al. (Eds.): TLDKS XXXIV, LNCS 10620, pp. 65–91, 2017.
https://doi.org/10.1007/978-3-662-55947-5_4

and domains' experts can build common taxonomies and ontologies in an easy way using P2PSW.

P2PSW combine the advantages of P2P wikis and the semantic wikis [2]. The replication of the semantic wiki pages in a distributed network enhances the performance, scalability, and fault-tolerance. The integration of the semantic aspect in P2PSW, improves the navigation, the search, and the knowledge extraction in the wikis. The semantic annotations in the wiki pages can be processed automatically by machines and they are exploited by semantic queries. P2PSW were first distributed on unstructured P2P networks, a recent work [3] proposed P2PSW distributed on structured P2P networks.

A P2P semantic wiki is a P2P network of autonomous semantic wiki servers (called also peers or nodes) that can dynamically join and leave the network. Every peer of a P2PSW hosts a copy of semantic wiki pages and a store for the semantic annotations extracted from these pages. 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. A semantic wiki page is an ordinary wiki page that contains semantic annotations. It can be seen as an ordered sequence of lines. The semantic annotations can be written as typed links. For instance, a semantic wiki page about "Jaguar" could be written as shown in Fig. 1.

```
Jaguar is a Native American word means "he who kills with one blow".
It is the third biggest cat behind the [isBiggest::Tiger].
Jaguar has many colors such as [hasColor:=Brown] one.
[category::Animal]
```

Fig. 1. Semantic Wiki Page about Jaguar

It contains four lines and three semantic annotations [isBiggest::Tiger], [hasColor:= Brown], and [category::Animal] about "Jaguar". Text and semantic annotations are stored in separate persistent storages. Text can be stored in files or a database. The semantic annotations are mapped into RDF statements where the subject is the page name. For example, the [isBiggest::Tiger] annotation will be stored as <"Jaguar", "isBiggest", "Tiger">. These annotations are stored in the peer triple store separate from text since relational database is not an ideal type of storage for semantic data. An RDF triple store organizes information in graphs rather than in fixed database tables. It is designed to answer queries in the SPARQL query language and to provide reasoning features on the ontological elements they store.

P2PSW are based on an optimistic replication model [4]. When a peer updates its local replica of a semantic wiki page, the replicas of the peers diverge. An update of a replica generates the corresponding operations i.e. insert or delete a line. An operation is processed in four steps: it is executed immediately against the local replica of the peer, broadcasted through the P2P network to other

peers, received by the other peers and integrated to their local replica. P2PSW use an optimistic synchronization algorithm to integrate the changes represented by operations and eventually ensure syntactic consistency. After integrating the same operations, wikis pages of the peers and their semantic annotations stores will become identical. The convergence of replicas is reached while preserving the execution order of the operations, and their intention independently of the concurrency. Each time the inserted or deleted line contains annotations, these annotations are extracted from the line, transformed into RDF statements and the local RDF triple store of the peer is updated. So, the merge of changes in P2PSW is made automatically by the synchronization algorithm and not by the users.

The first generation of P2PSW focused on ensuring syntactic convergence. They do not take in consideration the semantic consistency aspect of their content. While the syntactic consistency ensures that the semantic wiki pages of the peers and their stores will converge when integrating the same changes otherwise they diverge. The semantic consistency will be concerned with the consistency of the annotations in the semantic wiki pages of the peers. In other words, it will focus on ensuring that the common understanding of the users about the annotations is respected. The semantic consistency is not defined in the current P2PSWs. Current P2PSW do not support a mechanism to check the semantic consistency of the annotations in the semantic wiki pages. A user on a peer is not able to detect whether the annotations in a semantic wiki page are consistent or not. There is difference between the syntactic consistency and the semantic consistency, we explain it by running an example.

Consider two sites Site1 and Site2 replicating a semantic wiki page about "Jaguar", the page could be referring to a car for someone and an animal for another. Initially, the wiki page contains one line and is the same on both sites as shown in the Fig. 2. Suppose that a user on Site1 inserts the line "[category::Car]" at position2. Concurrently, a user on Site2 inserts the line "[category::Animal]" at the same position. The change on Site1 generates op1 = insert ("[category::Car]", 2) and the change on Site2 generates op2 = insert ("[category::Animal]", 2). The two operations are integrated locally, broadcasted through the network and eventually integrated on both sites. In P2PSW, the optimistic replication algorithm integrates op1 and op2 as follows. On Site1, first it inserts "[category::Car]" between line at position 1 and the end line of the page. When op2 is received on Site1, op2 specifies that "[category::Animal]" must be inserted between the same positions. The replication algorithm serializes op1 and op2 to make the operations commute and consequently to ensure convergence of the replicas on both sites. The Woot replication algorithm [5] uses the site identifiers in the synchronization which are unique and ordered. op2 is received from Site2 having an identifier greater than Site1, then "[category::Animal]" will be inserted after "[category::Car]". The same processing is made on Site2 and "[category::Car]" will be inserted before "[category::Animal]". The Logoot replication [6] generates a unique position between line 1 and the end line for line "[category::Car]" on Site1 and another unique position for "[category::Animal]" on Site2.

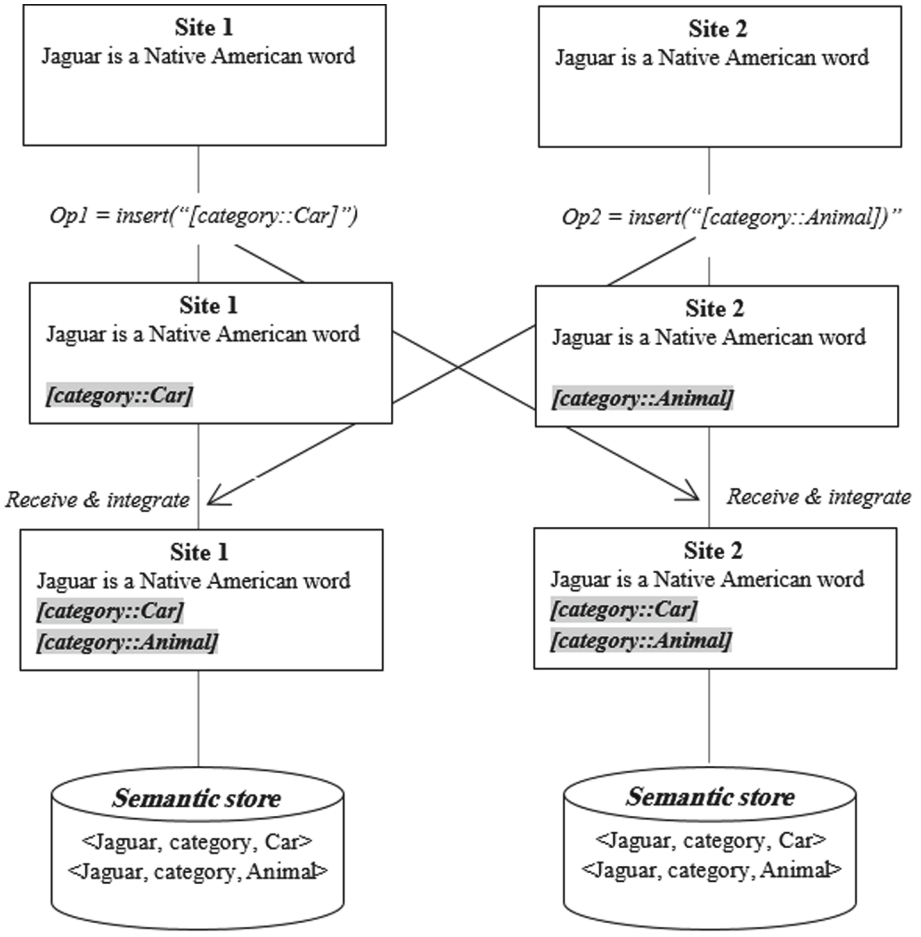

Fig. 2. Concurrent edition in P2PSW

Consequently, the lines "[category::Car]" and "[category::Animal]" will be inserted in the same order on both sites. The final result of an optimistic replication algorithm in P2PSW ensures that both sites ensure that the lines are inserted in the same order in the wiki pages and the triple stores contain the same semantic annotations as shown in Fig. 2. Both sites are syntactically convergent. However, a jaguar cannot have two disjoint categories Animal and Car at the same time. This statement cannot be made in the first generation of P2PSW. Thus, the obtained result is semantically inconsistent. Based on this result, running a semantic query to classify jaguars based on their category will return an erroneous result.

In the current P2PSWs, the result of the automatic merge could be anything and does not take in consideration the semantic consistency. There is no mech-

anism that helps the users to specify this constraint that the Animal and Car categories are distinct. The following annotations added by mistake [name:=3.2], [size:= −13], [birthdate:=2017-02-31], [brother::Apple Inc.] and the page is about a human and Apple is a company are examples about semantically inconsistent annotations that can be found in a semantic wiki page of a peer.

In spite the important role that play the semantic annotations in the P2PSW, working with semantically inconsistent annotations may lead to a loss of those benefits. Determining whether a semantic wiki page is semantically consistent requires checking the semantic consistency of its semantic annotations. This is not currently available in P2PSW. The annotations of a wiki page can be considered semantically consistent if they satisfy all the semantic rules on which they apply. For instance, a semantic rule will specify that a semantic wiki page should not belong to two or more disjoint categories.

The goal of this research work is to provide P2PSW with a generic semantic inconsistency detection mechanism (SIDM) that detects and proposes a solution for semantic inconsistency in P2PSW. SIDM will indicate not only the existence of the semantic inconsistency in the wiki pages but also the violated semantic rules. In addition, the inconsistency among the pages will also be detected. The semantic inconsistency is presented using a visualization mechanism we developed. **The outcome of this work is a second generation of P2PSW with an enhanced quality of content and knowledge.** Obviously, ensuring that the semantic annotations are semantically consistent has a major effect on the functionality of the P2PSW. It will improve the quality of the structured data in the different peers of the P2PSW.

The work focused on the design and the build of the semantic inconsistency detection mechanism. It includes defining semantic consistency in the context of P2PSW and the semantic rules (i.e. constraints) that determine the consistency of the annotations in the semantic wiki pages. The appropriate interfaces are created to write the semantic rules as special semantic wiki pages. The algorithm for the detection of the semantic inconsistency is developed. It detects the inconsistency existence and the inconsistent annotations. How the SIDM can be integrated in P2PSW and the semantic inconsistency is visualized follow.

The rest of the paper is organized as follows: Sect. 2 presents a state of art about the first generation of P2PSW. Section 3 discusses related work. Section 4 details the proposal that includes the algorithm, and the architecture. It describes the main steps followed in the development of the semantic inconsistency detection mechanism for P2PSW. Section 5 describes how to integrate SIDM in P2PSWs. Section 6 shows how the entire approach works. The last section concludes the paper and points to future work.

2 Peer-to-Peer Semantic Wikis

P2P Semantic Wikis constitute a collaborative editing tool for knowledge and ontology creation, share and management. They combine the advantages of semantic wikis and P2P wikis, also their technologies. They ensure a massive

collaboration in a distributed manner by replicating their data composed of semantic wikis pages and semantic annotations. Their main focus was to ensure the syntactic consistency of their replicated data.

P2PSW as a wiki constitute an easy to use collaborative editor for any type of users who aim to collaborate and to produce data and knowledge in a simple manner. The number of peers in that wiki can be huge, it is variable and can grow to thousands of thousands. Since these wikis were designed for mass collaboration they can generate huge amount of data called nowadays big data by very large number of users on the peers. The users on the peers can create as many as they want of semantic wiki pages. These pages will be replicated and their annotations are stored in triple stores that can handle billion of RDF triples. The users of these wikis can be researchers and professors or experts in ontology and taxonomy building or people with no experience in Semantic Web such as business managers, students, etc. The data in P2PSW can be reused later on as a reference or as a part in other projects or systems. Data are locally stored at the users' side and not on some distant servers owned by private companies.

The semantic aspect in P2PSW improves the organization and the extraction of knowledge from these data. In addition it enables users to produce a common understanding and vocabulary. The linked data represented as semantic annotations in the wiki pages are actually manipulated in P2PSW using the Semantic Web technologies. The annotations are translated into RDF triples, extracted and stored in the triple stores via SPARQL query language [14].

What distinguishes P2PSW from any other collaborative ontologies/knowledge editors in the Semantic Web is the real nature of collaborative editing. Many users can edit the wiki at the same time. Concurrent editions are handled and changes are merged automatically. It is not only about sharing or indexing the knowledge as other tools are limited to. There are two ways to build a P2PSW either by integrating the Semantic Web technologies in a P2P wiki or by distributing the architecture of a Semantic Wiki. Two P2PSW were developed SWOOKI and DSMW. SWOOKI followed the first way while DSMW adopted the second one. Both are based on an instantiation of the optimistic replication model in the context of semantic wikis. They ensure the CCI consistency model (Causality, Convergence, Intention) [4] of the replicated data. Next, I will briefly present these two P2PSW.

2.1 SWOOKI

SWOOKI [9] is the first P2P semantic wiki. A SWOOKI network is a set of interconnected semantic wiki servers. Each server hosts a replica of semantically annotated wiki pages and a triple store. It addresses specifically the problems of scalability and fault tolerance. SWOOKI adopts a total replication of the data on every peer of the network. Each peer can join and leave the network at any time. The produced knowledge can be searched, queried and extracted locally on each peer. SWOOKI uses Woot [5] as an optimistic replication mechanism to maintain the syntactic consistency of the replicated wiki pages and the replicated RDF repositories i.e. their convergence. It ensures the CCI consistency model.

SWOOKI was implemented in Java under the GPL license. You can download and test it last release 0.9 at http://sourceforge.net/projects/wooki.

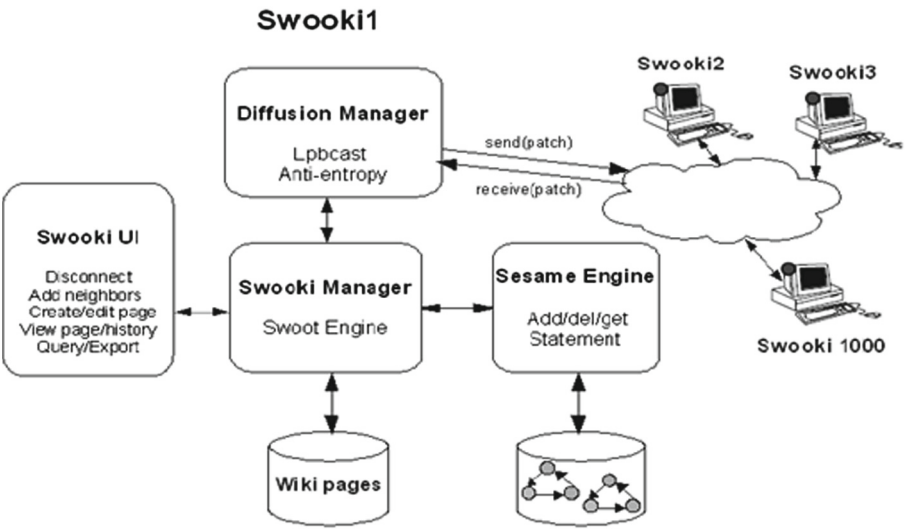

Fig. 3. Swooki architecture

SWOOKI Architecture. A SWOOKI server is composed of the following components (see Fig. 3):

- **User Interface:** The SWOOKI user interface (UI) component (see Fig. 4) is basically a regular wiki editor. It allows users to edit a view of a page by getting the page from the SWOOKI manager. Users can disconnect their peer to work in an off-line mode. They can add new neighbors in their list to work with. The UI allows users to see the history of a page, to search for pages having some annotation, to execute semantic queries, and to export the semantic annotations of the wiki pages in an RDF format.
- **SWOOKI Manager:** The SWOOKI manager is responsible for the generation and the integration of the editing patches which are sets of insert/delete operations. It implements the Woot algorithm. Its main method is to integrate all operations contained in the patch. Requesting and modifying a page or resolving a semantic query in the RDF repository pass through this manager.
- **Sesame Engine:** Sesame 2.0 is the RDF repository used in SWOOKI. Sesame is controlled by the SWOOKI manager for storing and retrieving RDF triples. This component allows also generating dynamic content for wiki pages using queries embedded in the wiki pages. It provides also a feature to export RDF graphs.
- **Diffusion Manager:** The diffusion manager is in charge to maintain the membership of the unstructured network and to implement a reliable broadcast.

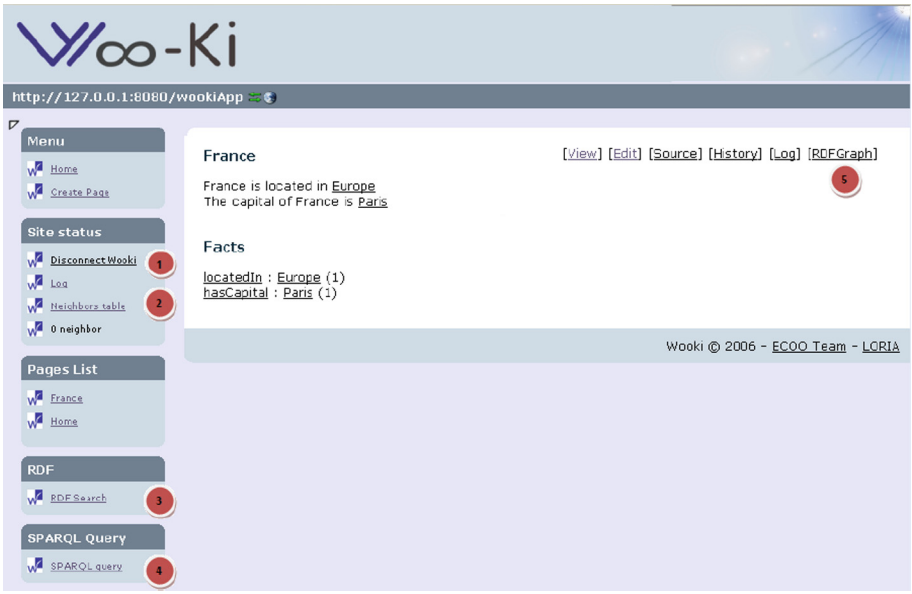

Fig. 4. Swooki user interface

2.2 Distributed Semantic Media Wiki (DSMW)

DSMW [10] is the second developed P2P Semantic Wiki. It allows to build a semantic Friend-to-Friend social network and to support multiple collaborative editing processes. In DSMW, a new model of collaboration called Push/Pull was developed. It is based on the notion of feeds. The idea was inspired from the work in Distributed Version Control Systems like Git. A generic ontology that covers the semantic wikis pages and their annotations, the changes and their history was proposed. Every DSMW element is an instantiation of this ontology and can be exploited semantically. DSMW is also based on an optimistic replication for the semantic wiki pages. It uses Logoot [6] for synchronization and to ensure the syntactic convergence of the replicated data.

DSWM allows users to build their own cooperation networks, every user declares explicitly with whom he would like to cooperate. Every user can have a DSMW server installed on his machine. He can create and edit his own semantic wiki pages as in a normal semantic wiki system. Later, he can decide to share or not these semantic wiki pages and decide with whom to share. The replication of data and the communication between servers is made through channels (push/pull feeds). These channels contain the changes made in the semantic wiki pages that can be shared and exchanged among peers. They are implemented as special semantic wiki pages.

When a semantic wiki page is updated on a multi-synchronous semantic wiki server, it generates a corresponding operation. This operation is processed in four steps: (1) it is executed immediately against the page, (2) published locally to the corresponding channels, (3) pulled remotely by authorized servers,

and (4) integrated to their local replica of the page. If needed, the integration process merges this modification with concurrent ones, generated either locally or received from a remote server. DSMW was implemented as an extension of Semantic MediaWiki which is also an extension of Wikipedia's wiki engine. The latest version of DSMW 1.2 can be downloaded and tested at http://momo54.github.io/DSMW.

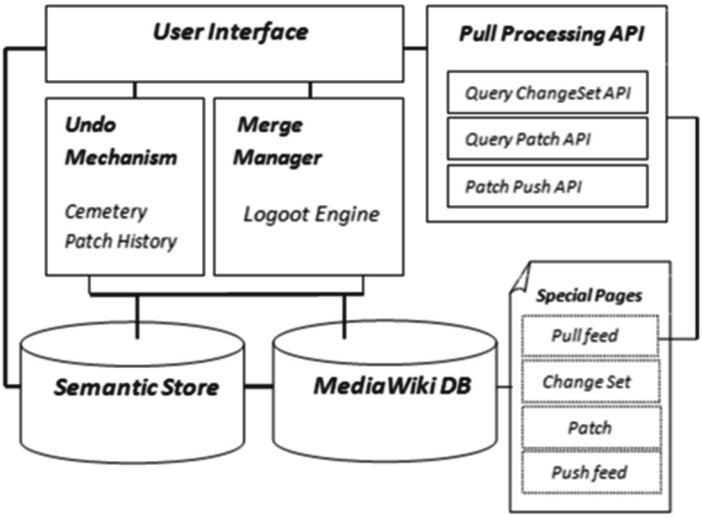

Fig. 5. DSMW architecture

DSMW Architecture. The DSMW architecture is illustrated in Fig. 5. Its components are given below:

- **User interface (UI):** Each semantic wiki page is associated with a special page (see Fig. 6) that shows the patches (i.e. the set of operations) integrated on that page, and the pushfeed to which it belongs.
- **Merge Manager:** is in charge of the integration of the operations. It synchronizes automatically the changes by implementing Logoot.
- **Diffusion manager:** This component is responsible for the generation and the propagation of the operations that represent the changes.
- **Data storage:** This component is constituted of a database that stores separately the semantic wiki pages and their annotations. It contains different namespaces (PullFeed, PushFeed, ChangeSet, Patch, and Operation) to separate the semantic wiki pages from the special ones.
- **Undo mechanism:** This component allows undoing changes at any time.

Fig. 6. DSMW page associated to 'Hello' page

3 Related Work

Semantic wikis are wiki engines that use the technologies of the Semantic Web to embed formalized knowledge in the wiki pages. This knowledge can be used to enhance the search and the navigation in the wiki or to update content dynamically. Some semantic wikis are dedicated to editing ontologies cooperatively such as Platypus, Rise, WikiSar, AceWiki, OntoWiki, and BOWiki. Others use ontologies as a reference for annotating wiki content such as IkeWiki, SWiM, and SweetWiki. Thus, they require to load/use a background ontology in the wiki. This provides guidance to the users during the annotations by proposing only valid completions. Semantic MediaWiki merged both approaches. First, it allowed users to add various types of ontological information to the wiki and to export these information later on. Then, it was extended by enabling importing ontologies in the wiki by authorized users. Based on the ontology, the system offers automatic classification of articles and supports the user in editing the wiki knowledge base. Many of the semantic wikis implement RDF and RDFs layer. Their vocabulary is not domain specific and thus does not allow to infer about domain specific relations.

Most of the semantic wikis do not check the consistency of the ontology they produce and do not support a reasoning feature [7]. Reasoning engine can perform consistency checks and derive additional implicit knowledge from the facts entered into the system. It uses predefined or user-defined rules in the knowledge base. Although reasoning is an important feature, it is only supported by a small number of wikis. The reasons for this might be that it is time-consuming, memory intensive, and can yield results that are not expected and/or traceable by the user. A reasoner may require significant resources (both in terms of processing and memory) which could slow down the wiki considerably. Consequently, it cannot be used to check the consistency of the P2PSW content which are designed for massive collaboration.

Some of the semantic wikis export the annotations or the ontologies defined in the wiki and check them by an external reasoning engine. Semantic MediaWiki uses external reasoner KAON2 to reason with the ontology it imports. This offers an automatic classification of its articles by adding a category to articles inferred from the ontology. It aims also supporting the users in editing the wiki knowledge base in a logically consistent manner. The users will be warned about inconsistencies in the wiki knowledge detected by the reasoner.

Semantic wikis are centralized based on servers. In case of concurrent changes in a semantic wiki, a conflict in the edition is presented to the user. It is up to him to solve the conflict and to make the merge manually or undoing its changes. The save of changes generates a new version of the wiki page. In semantic wikis, handling a semantic inconsistency is possible by either preventing the save of inconsistent semantic wiki pages or by checking the consistency of the saved wiki page modified by a user. An example of these wikis is BOWiki [8] that is destinated for collaborative editing of biomedical ontologies and gene data. It uses an OWL ontology with a description logic reasoner in order to perform consistency checks and queries. It evaluates newly entered data using an ontology in OWL-DL format. Only consistent semantic data will be stored in its semantic store. If an inconsistency is detected, the edited page is rejected with an explanation of the inconsistency. The use of these wikis could be hard for some users and the way they handle semantic inconsistency cannot be applied in the context of P2PSW. In P2PSW, the semantic consistency of remote changes can be only checked after they are received and integrated since the merge is automatically made by the wiki and not by the users.

4 Semantic Inconsistency Detection Mechanism

This section describes the proposal and is structured as follows. First the semantic consistency in the context of P2PSW and the semantic consistency rules are defined. Then the semantic inconsistency detection approach and the developed algorithm are presented.

4.1 Semantic Consistency in Peer-to-Peer Semantic Wikis

I define the semantic consistency in P2PSW as follows: A P2PSW is semantically consistent if all its semantic wiki pages are semantically consistent. A semantic wiki page is semantically consistent if its all semantic annotations are semantically consistent. The semantic annotations of a wiki page are semantically consistent when they are in a state in which all the semantic rules concerning these annotations are satisfied. Otherwise, the annotations are considered semantically inconsistent also their pages. So, what actually determines the consistency of the annotations are the semantic rules. In the absence of those rules, no information can be given about the consistency of the annotations.

First, we studied the existent constraints and restrictions that can be expressed in RDF schemas (RDFS) and OWL and then we derived the consistency rules that cover these. The constraints can be extended easily if necessary.

They can be expressed as semantic annotations and their satisfaction can be checked using SPARQL queries. Many research works on semantic consistency checking such in [12,13] focused on defining axioms and checking their consistency. In our approach, we followed the same principle.

In ontological terms, the annotations in a wiki page could represent a part of the ABox of an ontology i.e. the sets of assertions about the individuals which are instances of concepts. The semantic rules we define will represent the TBox of an ontology i.e. the set of concepts and their properties to formally describe a domain. In our consistency check of the P2PSW content, we are interested in only one major task of a reasoner which is checking the consistency of ABox with respect to TBox i.e. determining whether individuals in ABox do not violate axioms described by TBox.

Before detailing the semantic rules and their use, I will explain how the annotations in a semantic wiki page will be extracted, mapped into RDF triples and stored in the triple stores in the second generation of P2PSWs. I clarify three terms: instance, property, and concept in the context of P2PSW:

1. **Instance:** Every semantic wiki page that contains the annotation [category::Concept] is an instance of the Concept. Like in Object Oriented Programming, an instance can be seen as an object of the class Concept. A semantic wiki page can be an instance of many concepts, i.e. contains many annotations $[\text{category::Concept}_i]_{i=1,..,n}$.
2. **Property:** In semantic wiki pages, a property or predicate describes a relation between the semantic wiki page and another page or a characteristic of that page. It can be written as [property1::SemWikiPage2] or [property1:=Value].
3. **Concept:** Concepts are classes that provide an abstraction mechanism for grouping with similar characteristics. A concept can be seen as a category used to group a set of semantic wiki pages.

Example. We illustrate the three previous terms through the example given below:

```
Jaguar is the third biggest cat behind the [isBiggest::Tiger].
It has many colors but they are beautiful with [hasColor:=Darkest] one.
It has [hasLegs:=4] legs and runs very fast.
[category::Animal]
```

1. The semantic wiki page "Jaguar" is an instance of Animal. This annotation is mapped into an RDF triple and stored in the triple store as follows:
   <Jaguar> <rdf:type> <Concepts/Animal>
2. The properties of "Jaguar" page are isBiggest, hasColor, and hasLegs. They are stored in the triple store with the corresponding objects as follows:
   <Jaguar><Properties/isBiggest><Tiger>
   <Jaguar><Properties/hasColor>"Darkest"
   <Jaguar><Properties/hasLegs>"4"
3. There is the "Animal" concept to which belongs the "Jaguar" instance.

4.2 Semantic Consistency Rules

I define semantic consistency rules in P2PSW as the constraints that can applied on the semantic annotations. They are similar to integrity constraints applied to data in databases. These rules represent the constraints defined on the properties and the concepts. The rules on properties concern the domain and the range of the Properties, while the rules on concepts concern the cardinality of the properties in a Concept and the relations between Concepts. To be integrated in P2PSWs, first the properties and the concepts pages should be created. Then the annotations that represent the constraints are inserted in these pages.

Semantic Consistency Rule on Properties. I decided to use *domain* and *range* as semantic consistency rules on properties as shown in Table 1. I borrowed the idea from RDF Schema that uses them to associate constraints to properties. *rdfs:domain* and *rdfs:range* allow making statements about the contexts in which certain properties "make sense". The role of these constraints is:

- **rdfs:range** is used to constrain property values.
- **rdfs:domain** is used to specify a class on which a property may be used.

To define these constraints on a property, we create a special semantic wiki page for that property. The namespace *Properties* will be used for all the property pages.

Table 1. Property constraints

Constraints on properties	
P2PSW annotations	RDF
[domain::URI]	rdfs:domain
[range::URI |Literal]	rdfs:range

We insert the annotations [domain::URI] ([range::URI |Literal] respectively) in the semantic wiki page property1 to specify that property1 has a domain URI (has range a URI or a literal respectively). Each annotation has its equivalent in RDF as shown in Table 1. When saved, the annotations of the property1 are updated in the triple store. First, the triples of that property are removed from the store and then the annotations of the saved page are mapped into RDF triples and stored in the triple store.

We map these annotations into RDF triples as shown below:

<Properties/property1><rdfs:domain><Concepts/URI>
<Properties/property1><rdfs:range><Concepts/URI> Or
<Properties/property1><rdfs:range>
<http://www.w3.org/2000/01/rdf-schema#Literal>.

In fact, the range of a property could be an URI i.e. some concept or a literal i.e. one of the datatypes: an integer, a float, a boolean, a string, a symbol, etc.

Example of Constraints Definition on Properties. To define the domain and range of isBiggest property, we create a semantic wiki page "isBiggest" in the *Properties* namespace and insert the appropriate annotations as illustrated in the following text:

```
[domain::Animal]
[range::Cat]
```

Semantic Consistency Rule on Concepts. I define two types of semantic consistency rules on a concept: (1) the cardinality of the concept properties and (2) the relations between this concept and other concepts. To define these constraints on a concept, a user on a peer must create a special semantic wiki page for that concept. The namespace *Concepts* will be used for all the concept pages. I defined the semantic annotations that can be added in the concepts. They express the constraints that can be applied on concepts. These annotations and their equivalence in OWL language [11] are given in Table 2. Actually, cardinality constraints can be used to make a property required (at least one), to allow only a specific number of values for that property, or to insist that a property must not occur. OWL provides three constructs for restricting the cardinality of properties locally within a class context. $owl:minCardinality$, $owl:maxCardinality$, and $owl:cardinality$ describe a class of all individuals that have at least N, at most N, and exactly N semantically distinct values for the concerned property, where N is the value of the cardinality constraint. On the other hand, C1 $rdfs:subClassOf$ C2, C1 $owl:equivalentClass$ C2, or C1 $owl:disjointWith$ C2 allow to say that the set of instances of C1 is a subset, the same, or has no instance in common with the set of instances of C2, where C1 and C2 are two concepts. I map the annotations inserted in concepts into RDF triples as shown below:

<Concepts/concept1>	<owl:equivalentClass>	<Concepts/URI>
<Concepts/concept1>	<owl:disjointWith>	<Concepts/URI>
<Concepts/concept1>	<rdfs:subClassOf>	<Concepts/URI>
<Concepts/concept1>	<property>	"value:max"
<Concepts/concept1>	<property>	"value:min"
<Concepts/concept1>	<property>	"value:exactly"

When saved, the annotations i.e. the constraints of the concept1 page are inserted in the triple store.

Example of constraints definition on concepts. We can define an Animal concept as a class that has at least one color, at least two legs, and one isBiggest property. We can say also that an Animal is not a Car. To do so, we create a semantic wiki page "Animal" with the following annotations.

Table 2. Concept constraints

Constraints on concepts	
P2PSW annotations	OWL
[equivalent::URI]	owl:equivalentClass
[disjoint::URI]	owl:disjointWith
[property:min=value]	owl:minCardinality
[property:max=value]	owl:maxCardinality
[property:exactly=value]	owl:cardinality
[subClass:URI]	rdfs:subClassOf

```
[hasLegs:min=2]       [hasColor:min=1]
[disjoint::Car]    [isBiggest:exactly=1]
```

4.3 Semantic Inconsistency Detection Approach

The semantic inconsistency detection approach I developed is made of many components shown in the Fig. 7. The semantic inconsistency checker detects inconsistency on three levels: the semantic wiki page, the concept, and the property level. A user can check whether a page is consistent or select a property or concept to check in order to identify if there are semantic wiki pages that violate it. The checker works by running SPARQL queries [14] on the triple store of the P2PSW peer. The result of the query is displayed using a visualization mechanism that shows the inconsistency when it exists. In this section, I describe the possible inconsistencies that can take place, how they are detected on every level, and the developed algorithm.

Check Consistency on Semantic Wiki Page Level. To check the consistency of a semantic wiki page (SWP), we check the satisfaction of the semantic consistency rules on concepts and properties associated with the semantic annotations of that page. A semantic inconsistency occurs when one or many semantic consistency rules are violated. I consider that there is no contradiction in the semantic consistency rules definition. In addition, checking the inconsistency by a user on a peer can be made at any time. The inconsistency detection is made via SPARQL queries since both the semantic annotations and the semantic consistency rules are stored as RDF triples in the triple store of the peer. To detect the semantic inconsistency in a semantic wiki page SWP, we follow these steps:

1. Select all the semantic annotations in SWP. If there are no annotations in the result then there is nothing to check. We consider a semantic wiki page without annotations as a semantically consistent one. Otherwise, go to step2.
2. Check the satisfaction of the semantic consistency rules on the concepts of SWP. Select the concepts, let SC be the set of these concepts. Two cases exist:

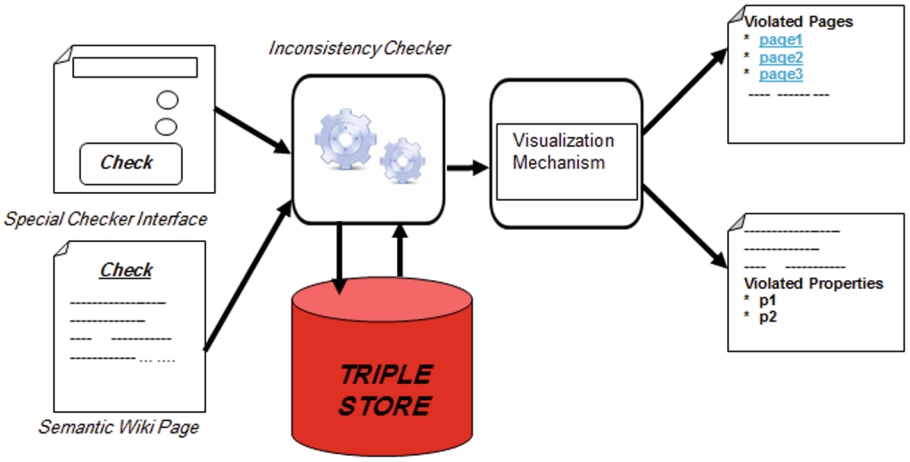

Fig. 7. The semantic consistency checker components

2.1. There is no concept found, SC is empty, i.e. SWP does not belong to any concept. In this case, check the satisfaction of the semantic rules on the properties of SWP, go to step 5.
2.2. SC is not empty, SC = $\{C_1, C_2, \ldots, C_n\}$. Check the disjoint constraints on these concepts go to step 3.

3. Compute SDC the set of disjoint concepts in SC. Two cases exist:
3.1. There are disjoint concepts, i.e. SDC = $\{C_k\}_{1<=k<=n}$. Display SWP is semantically inconsistent and the properties of every concept in SDC. The semantic inconsistency checker stops.
3.2. There is no disjoint concept, SDC is empty. Check the constraints on every concept C in SC, go to step 4.

4. For each concept C in SC, select the properties SP in C.
For each property P in SP:
4.1. Check the cardinality constraint of P. It consists of two steps:
4.1.1. Compute the cardinality *cp* of P in SWP i.e. the number of times P is present in SWP with different values.
4.1.2. Compare *cp* with the cardinality constraints on P in the concept C. If (*cp* < minCardinality) or (*cp* > maxCardinality) or (*cp* <> cardinality) then Display semantic inconsistency on the cardinality of P in C.
4.2. Check the range constraint of P. It consists of two steps:
4.2.1. Compute the range R of P in SWP.
– If the *value* of P i.e. the object in SWP is another semantic wiki page SWP' then check if SWP' exists. If SWP' does not exist then there is no information whether the range R of that property P is violated or not. If SWP' exists, we compute the set of concepts (i.e. ranges) SR to which belongs SWP'.
– If the *value* of P in SWP is a literal, let us call it SR.

4.2.2 Compare SR with the range R" of P defined in the semantic wiki page of the property P only if SR is not empty. If R' <> SR then Display semantic inconsistency on the range of P.

5. Check the satisfaction of the semantic rules on the properties of SWP. Here we check the constraints on the properties that do not belong to any concept of SWP, i.e. only the unchecked properties. Let us call SUP the set of those properties.
   For each property P in SUP:
   5.1. Check the domain constraint on P. If the domain concept DC of P does not exist i.e. is not defined or the set of concepts SC in SWP is empty then there is no information else Display there is a possible semantic inconsistency of the domain of P in SWP.
   5.2. Check the range constraint of P, go to step 4.2.

Some choices we made in the SIDM are presented below:

1. The check stops when two or more disjoint concepts are found in a semantic wiki page. To facilitate solving the inconsistency, we display the concepts and their properties. Another alternative could be letting the algorithm to continue checking the semantic inconsistency of the properties in SWP that belong to the non-disjoint concepts and that do not belong to any concept.
2. We consider the existence of a *possible* semantic inconsistency when the domain of a property defined in the property page is different than the categories of the SWP where the property is used. For instance, consider that the domain of a property is Bike, and the categories of SWP are Vehicle and Bicycle. If we compare Bike with the categories, they are different; however Bike and Bicycle are the same. In the context of Semantic Web, two concepts are equivalent if there is an explicit statement stating so. In the context of P2PSW, we can compare the equivalence and the disjoint of the categories with the domain of the property to possibly detect the inconsistency if it exists. We decided to leave that to the users.
3. To ensure scalability, we can use the construct SPARQL queries to extract at once all the required information for the SIDM algorithm from the store. The result is an RDF graph that will be used to detect the inconsistency and there is no need to interrogate the triple store again multiple times.

Check Consistency on Concept Level. Four types of constraints can be defined in concept pages which are subclass, disjoint, equivalent, and the cardinality of the properties that could be present in the concept. The constraints definitions and their violation detection are presented as follows:

1. **Subclass constraint:** a concept C is a subclass of a concept C' if every instance of C is an instance of C'. The constraint is violated when an instance of C doesn't belong to C'. An instance is a semantic wiki page. In other words, the constraint is violated when there exists a semantic wiki page that belongs to the category C and not to C'.

2. **Disjoint constraint:** a concept C is disjoint with a concept C' if every instance of C is not an instance of C'. The semantic wiki pages that belong to disjoint categories with C will be extracted and displayed. In the semantic consistency checker, we check only the satisfaction of the rules [disjoint::C'] defined in the Concept C page. We do not consider the case [disjoint::C] defined in C'.
3. **Equivalent constraint:** Two concepts C and C' are equivalent if they have the same instance set called a class extension. The constraint is violated when a semantic wiki page belongs to C and not to C'.
4. **Cardinality constraint:** Three types of constraints can be defined on the properties cardinality in a concept C. The constraints on the properties cardinality can be minimum cardinality, maximum cardinality, and exact cardinality. The checker detects the satisfaction or the violation of these constraints. We defined a function called checkCardinality() that takes as parameter a concept C and returns the set of pages that contain a property of C with a violated constraint. The function works as follows: first we extract the properties present in the concept C, then for each one of them we extract the pages that belong to C violating the properties cardinality constraints defined in C. A detailed description of the checkCardinality function is given in Algorithm 1.

```
Function checkCardinality (Concept C)
    VPS ← {}; //set of violated pages ;
    PRS ← {prop ∈ Properties/ [prop:min=V] ∨ [prop:max=V'] ∨
     [prop:exactly=V"] ∈ C };
    if (PRS = {}) then
        return VPS;
    else
        PGS ← {page ∈ Pages / [category::C] ∈ page};
        if (PGS = {}) then
            return VPS;
        else
            for each page ∈ PGS do
                for each prop ∈ PRS do
                    cp ← Cardinality {prop, page};
                    if ((cp < V) ∨ (cp > V') ∨ (cp ≠ V'')) then
                        VPS = VPS ∪ {page};
                    end
                end
            end
            return VPS;
        end
    end
```

Algorithm 1. The checkCardinality function

Check Consistency on Property Level. In a property page, two types of constraints can be defined which are the domain and the range constraints of a property.

Domain constraint: This constraint specifies the concept that represents the domain of a property. This constraint is violated if the property belongs to a page that is an instance of one or many concepts disjoint with the domain of that property. We define the checkDomain( ) function that takes the property name as parameter and returns the pages that violate this constraint. A detailed description of the checkDomain function is given in Algorithm 2.

```
Function checkDomain (Property P)
    VPS ← {}; //set of violated pages ;
    D ← Domain(P);
    if (∄ D) then
        return VPS;
    else
        PGS ← {page ∈ Pages / [P::V] ∨ [P:=V'] ∈ page};
        if (PGS = {}) then
            return VPS;
        else
            for each page ∈ PGS do
                CS ← {C ∈ Concepts/ [category::C] ∈ page};
                if (CS = {}) then
                    continue;
                else
                    if (D ∉ CS) then
                        VPS = VPS ∪ {page};
                    end
                end
            end
            return VPS;
        end
    end
```

Algorithm 2. The checkDomain function

Range constraint: This constraint gives the range concept of a property P. It is violated when [P::v] or [P:=v] is found in the checked semantic wiki page and v is different than the defined range in P. If v is a literal, we compare directly the data type of v with the range. However, when v is a URI (a semantic wiki page), we compute the concepts of that page and compare them with the range. If they are different, then the page containing P violates the range constraint of P. We define the checkRange() function that computes the pages that violate this constraint. A detailed description of the checkRange function is given in Algorithm 3.

```
Function checkRange (Property P)
    VPS ← {}; //set of violated pages ;
    R ← Range(P);
    if (∄ R) then
        return VPS;
    else
        PGS ← {page ∈ Pages / [P::V] ∨ [P:=V'] ∈ page};
        if (PGS ≠ {}) then
            for each page ∈ PGS do
                PVS ← { V ∈ Values / [P::V] ∨ [P:=V'] ∈ page} ;
                if (PVS ≠ {}) then
                    for each V ∈ PVS do
                        if ( isLiteral(V) ∧ Range(V) ≠ R) then
                            VPS ← VPS ∪ {page} ;
                            break;
                        end
                        if (isURI(V)) then
                            CP ← { C ∈ Concepts/ [category::C] ∈ V};
                            if ((CP ≠ {}) ∧ (R ∉ CP)) then
                                VPS ← VPS ∪ {page} ;
                                break ;
                            end
                        end
                    end
                end
            end
            return VPS;
        end
    end
```

Algorithm 3. The checkRange function

The SIDM was implemented using PHP and JQuery as programming languages, WAMP as the Web server, and ARC2 as the triple store. The implementation is a simulation of a P2PSW peer. The prototype can be downloaded and tested at this address: https://sites.google.com/site/charbelrahhal/home/developed-softwares.

5 Integrate the Semantic Inconsistency Detection Mechanism in the First Generation of P2PSW

In this section, I present how the developed SIDM can be integrated in SWOOKI and DSMW. There are two possible cases, either the users on the peers build the semantic inconsistency rules incrementally or the set of rules is fixed and is the same on all the peers.

5.1 Variable Set of Semantic Consistency Rules

On every peer, the user can create and edit two types of special semantic wiki pages: concepts and properties. These pages will contain semantic annotations that represent the semantic consistency rules. Once the changes are saved, the pages will be replicated and the annotations will be extracted and stored in the triple store of the peer. These will be used as an input to the SIDM and later on to check the semantic inconsistency of the wiki pages on the user's peer. In this case, the set of semantic consistency rules will diverge on the peers and will be handled differently in SWOOKI and in DSMW.

- **In SWOOKI:** when the user specifies the semantic consistency rules locally in a concept or a property and saves. These will be integrated locally, propagated through the network, and integrated on the other peers. Hence, the concepts and the properties will be replicated on the peers (see Fig. 8). A user on a peer can check if there are changes occurred in the semantic consistency rules before running the SIDM. Either he agrees with these changes and starts the checker or he can undo them. Undoing changes exist in SWOOKI. Thanks to its optimistic replication algorithm, SWOOKI ensures that eventually after integrating all the changes, the semantic consistency rules will converge on the peers.

- **In DSMW:** every user can specify its semantic consistency rules and publishes them when he is ready. Other peers can create pull feeds and pull the rules specified by that user (see Fig. 9). The process of publishing and pulling among the peers can continue until an agreement is reached or stops when the users decide to. The SIDM can be run at any time. In case of an agreement, the semantic consistency rules will be same on the peers. DSMW also supports an undo mechanism. An advantage DSMW has over SWOOKI is that users can be aware when a change occurs in the pushfeed and pull it afterwards.

5.2 Same Set of Semantic Consistency Rules

Another alternative is to use a fixed set of semantic consistency rules on all the peers before to start running SIDM (see Fig. 10). First, the users on the peers will select the same specification/ ontology from a list. An ontology specifies the semantic consistency rules to be created. The list could be a special semantic wiki page or interface. It refers to a set of specifications that can be imported from different locations. Once the ontology is selected, the corresponding concept and property pages will be created with their semantic annotations. To ensure a fixed set of rules, these pages could not be directly editable; they are read only pages. Finally, the annotations are mapped into RDF statements and stored in the triple store. As a result, the semantic consistency rules will be the same on all the peers and the SIDM will have the same input everywhere. This process can apply on both SWOOKI and DSMW.

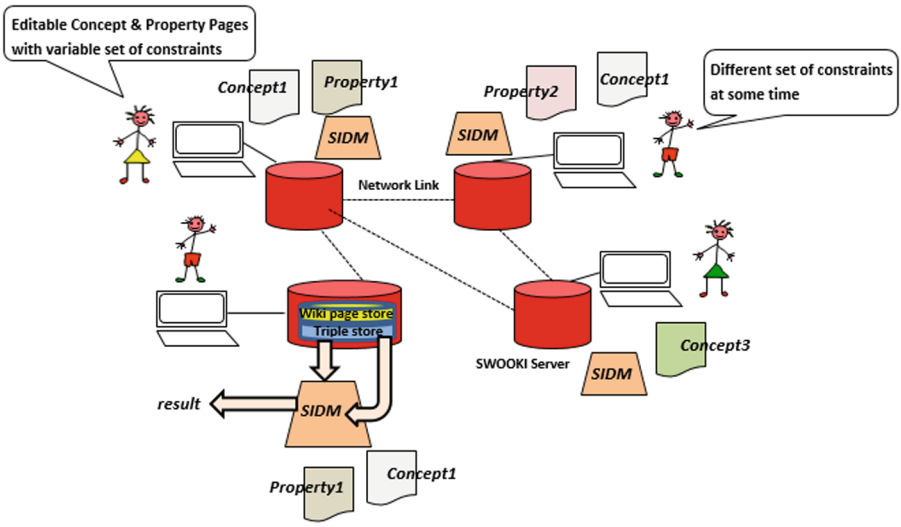

Fig. 8. Different semantic rules on SWOOKI

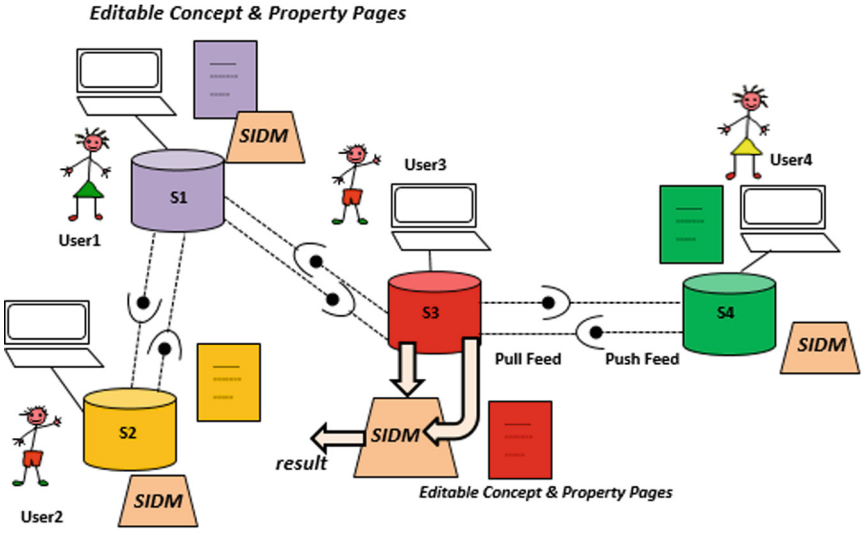

Fig. 9. Different semantic rules on DSMW

6 Running SIDM

This section presents two ways to check the semantic inconsistency either directly on a semantic wiki page or on a property/concept level. In the later one, it checks whether there are one or many semantic wiki pages that violate the constraints on a property or a concept.

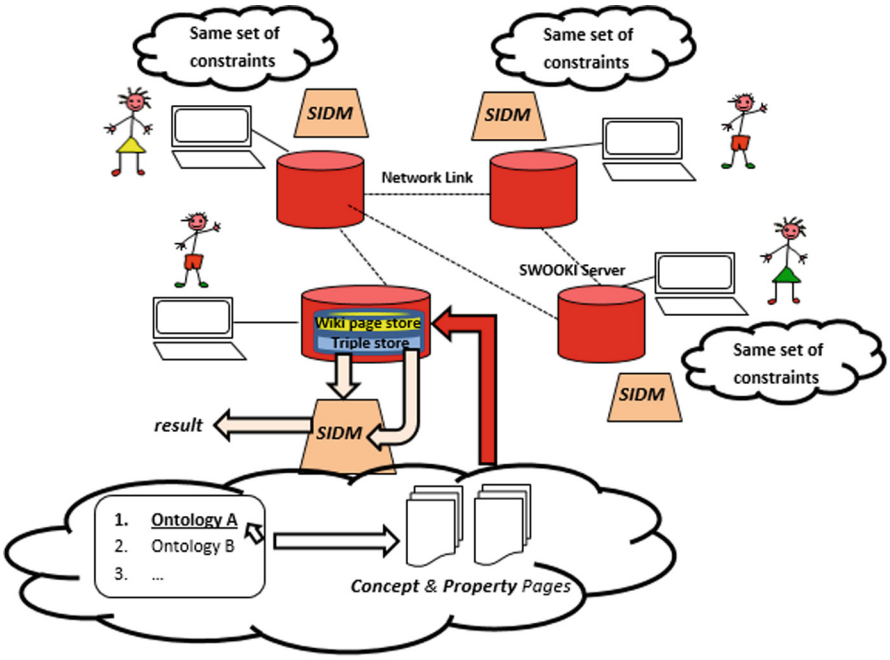

Fig. 10. Same set of semantic rules in SWOOKI

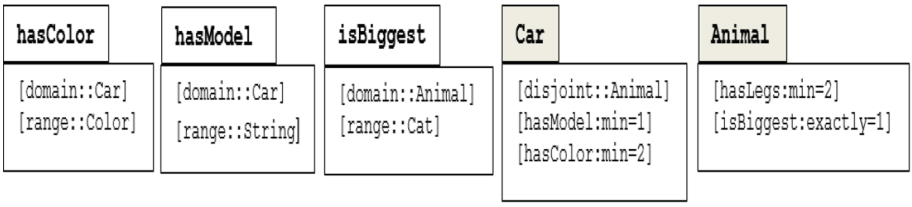

Fig. 11. Property and Concept Pages with their constraints

6.1 Check Consistency on a Semantic Wiki Page Level

Consider that in the P2PSW there are only one semantic wiki page "Jaguar", three property pages "hasColor", "hasModel" and "isBiggest", and two concept pages "Animal" and "Car". The property and concept pages are shown in Fig. 11. We want to check the semantic consistency of the "Jaguar" page. The annotations in "Jaguar" indicate that Jaguar is at the same time a car and an animal. This is could be obtained by the edition of "Jaguar" page on two different peers and the current wiki page content is the result of the automatic changes merge.

First, we click on "Check Consistency" tab (see Fig. 12) on the "Jaguar" page. When the tab is clicked, the SIDM is executed and the result is displayed in a check consistency page (see Fig. 13). It shows that the "Jaguar" page is inconsis-

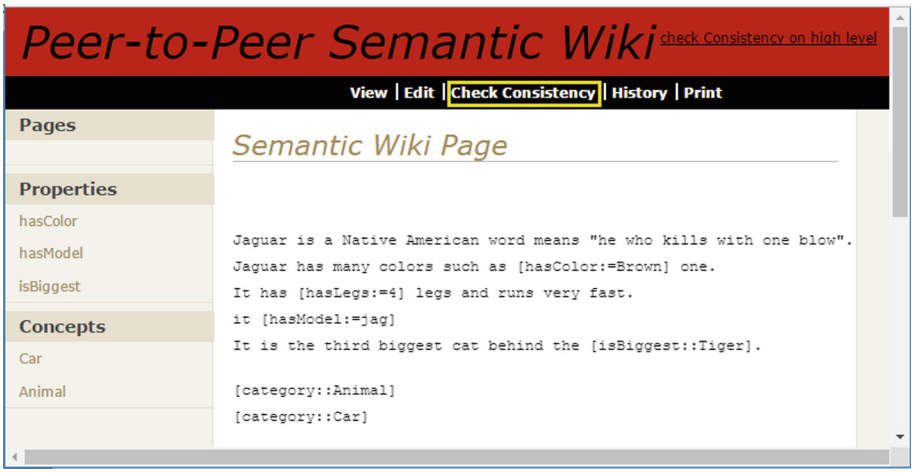

Fig. 12. Jaguar Semantic Wiki Page

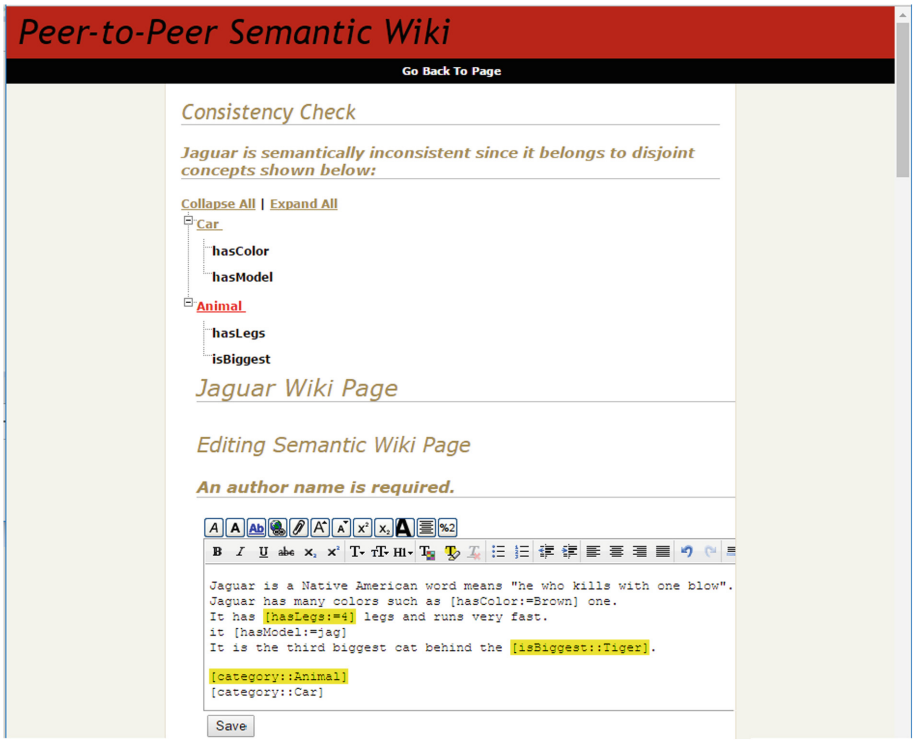

Fig. 13. Highlighted annotations in the check consistency page

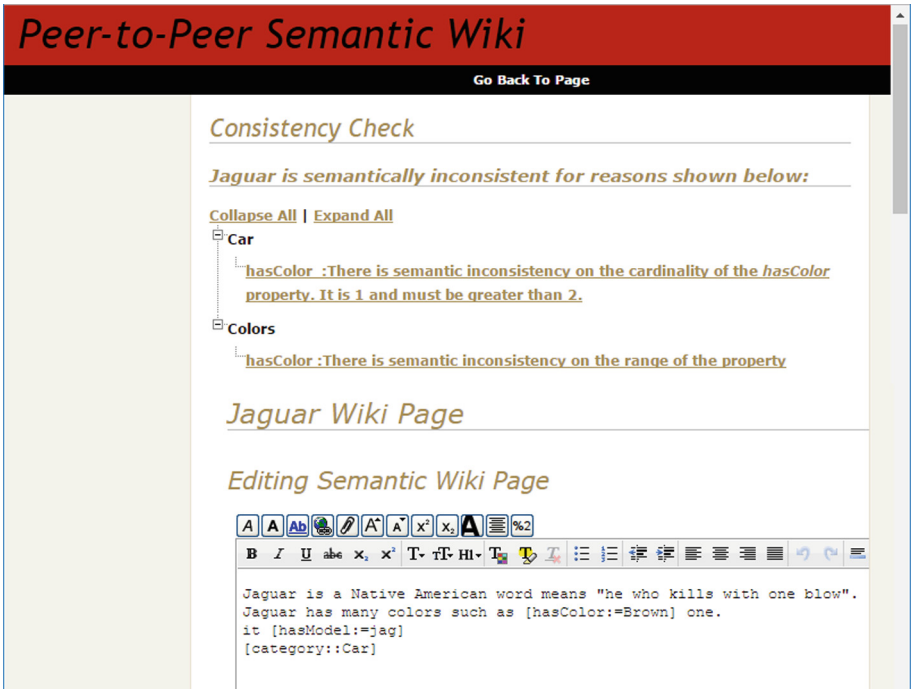

Fig. 14. The Jaguar page rechecked

tent since it belongs to two disjoint concepts Car and Animal. The inconsistent concepts are visualized via a treeview. The nodes in the treeview show these concepts along with their properties, they can be expanded or collapsed. We can choose to remove all the annotations in the page related to animal. By clicking on the Animal tree node, all the concerned annotations will be highlighted and can be easily removed. We can copy the deleted lines into another wiki page that can be called Jaguar Animal.

Another check of the page consistency (see Fig. 14) shows that the page is still inconsistent. It points to two types of inconsistency found in the page. The "Jaguar" page is a Car and has only one color property. However, the semantic consistency rules in Car Concept page specify that every car should have at least two colors. In addition, the range of hasColor property is a literal in the "Jaguar" page ([hasColor:=Brown]) which violates the semantic consistency rule in Color Property page that determines the range of hasColor property as an instance of a Color concept. In this case, we can make the necessary changes in the "Jaguar" page. A last check on the page will show that "Jaguar" is semantically consistent.

6.2 Check Consistency on a Concept/Property Level

We associated with every semantic wiki page a "Check Consistency on high level" link (see Fig. 12) which will open a special wiki page in the browser that looks like

Fig. 15. Check consistency high level on concepts/properties

Fig. 15. The "Check consistency high level" page contains two options: Concepts and properties. In this example, SIDM checks the semantic inconsistency of the concept Car. It will display the disjoint concepts with Car and the pages that contain these disjoint concepts via a treeview.

7 Conclusion

This section gives an evaluation of the approach and points to perspectives and future works. The research work conducted focused on building a second generation of P2PSW by providing them with a semantic inconsistency detection mechanism. The SIDM improves the quality of the structured data in P2PSW, and consequently their functionality and the knowledge extraction.

The development of SIDM followed many steps: (1) defining the semantic inconsistency in the context of P2PSW, (2) defining the semantic rules and the way they can be integrated in the wiki, and (3) developing an algorithm for the detection of the semantic inconsistency on different levels. As a result, we can detect the inconsistency of the entire P2PSW. The SIDM not only detects the inconsistency existence but also specifies the inconsistent annotations. At the end, SIDM was implemented and tests were ran to remove any bugs and optimize SIDM algorithm.

SIDM is designed for P2PSW but it can be integrated in any semantic wiki that manipulates the annotations as typed links such as Semantic MediaWiki. This is can be done easily since SIDM was implemented in PHP which is used in Semantic MediaWiki.

The complexity of the inconsistency detection algorithm depends on the number of the annotations in the wiki pages and on the number of semantic wiki pages checked at the same time. To detect the inconsistency, SIDM can extract first the required information from the store using a graph in one pass and make the check on it. It means that every check requires only one request to the triple store. Triple stores were designed to be very scalable. They can store billions of

triples, handle a large number of requests and answer them in very short time since they use different indexations. Currently, we are conducting user studies to evaluate our approach. These studies will help us to enhance the approach and the functionality in the P2PSW in general.

References

1. Skaf-Molli, H., Rahhal, C., Molli, P.: Peer-to-peer semantic wikis. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2009. LNCS, vol. 5690, pp. 196–213. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03573-9_16
2. Meilender, T., Jay, N., Lieber, J., Palomares, F.: Semantic wiki engines: a state of the art. Semantic Web J. (2010)
3. Rahhal, C., Yactin, H.: Semantic wikis distributed on structured peer-to-peer networks. In: CSCEET 2017: The Fourth International Conference on Computer Science, Computer Engineering, and Education Technologies, 26–28 April 2017
4. Saito, Y., Shapiro, M.: Optimistic replication. ACM Comput. Surv. **37**, 42–81 (2005). doi:10.1145/1057977.1057980
5. Oster, G., Urso, P., Molli, P., Imine, A.: Data consistency for P2P Collaborative editing. In CSCW'06: ACM Conference on Computer Supported Cooperative Work, pp. 259–268, 4–8 November (2006). doi:10.1145/1180875.1180916
6. Weiss, S., Urso, P., Molli, P.: Logoot: a scalable optimistic replication algorithm for collaborative editing on P2P networks. In: ICDCS 2009: 29th IEEE International Conference on Distributed Computing Systems, pp. 404–412, 22–26 June 2009. doi:10.1109/ICDCS.2009.75
7. Buffa, M., Gandon, F., Ereteo, G., Sander, P., Faron, C.: SweetWiki: a semantic wiki. J. Web Semant. **6**, 84–97 (2008). doi:10.1016/j.websem.2007.11.003
8. Hoehndorf, R., et al.: BOWiki: an ontology-based wiki for annotation of data and integration of knowledge in biology. J. BMC Bioinf. **10**, May 2009. doi:10.1186/1471-2105-10-S5-S5
9. Rahhal, C., Skaf-Molli, H., Molli, P.: Swooki, un wiki sémantique sur réseau pair-à-pair. Journal Ingénierie des Systèmes d'Information (ISI) **14**(1), 117–140 (2009). doi:10.3166/isi.14.1.117-140, Lavoisier
10. Rahhal, C., Skaf-Molli, H., Molli, P., Weiss, S.: Multi-synchronous collaborative semantic wikis. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 115–129. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04409-0_17
11. OWL working group: Web Ontology Language (OWL), 11 December 2012. https://www.w3.org/OWL,
12. Yang, S., Tan, H., Jinzhao, W.: Semantic Consistency Checking in Building Ontology from Heterogeneous Sources. J. Appl. Math. vol. 2014 (2014). doi:10.1155/2014/181938
13. Han, X., Sun, L.: Semantic consistency: a local subspace based method for distant supervised relation extraction. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, 22–27 June 2014
14. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C recommendation, 21 March 2013. https://www.w3.org/TR/sparql11-query/