Peermarks: P2P based Social Bookmarking

Giovanni Cortese, Luca Lorenz, Dominic Battré, Felix Heine, André Höing, and Odej Kao

2007
Peermarks: P2P based Social Bookmarking

Dominic Battré\textsuperscript{1}, Giovanni Cortese\textsuperscript{2}, Felix Heine\textsuperscript{1}, André Höing\textsuperscript{1}, Odej Kao\textsuperscript{1}

\textsuperscript{1}: Technical University of Berlin, Germany
\textsuperscript{2}: Radiolabs, Italy

\section*{Abstract.} In this paper we present Peermarks, an application which implements a social bookmarking service that we developed as a case study for our peer-to-peer XML/RDF storage and query technology. The application allows users to classify web resources in a bottom-up style (using tags) or to use a hierarchical categorization scheme. Peermarks supports both XML data storage and RDF-based inferencing. In Peermarks, users share bookmarks and also their classification knowledge. We discuss how the proposed system allows global classification knowledge to emerge from individual user categorizations.

\section{Introduction}
Social networking sites such as Flickr or Delicious are a very popular, new class of applications. More specifically, social bookmarking sites allow Internet users to store, organize, share and search bookmarks of web pages. In a social bookmarking system, users save links to web pages that they want to remember and/or share. Most social network sites today are implemented using a centralized storage architecture.

In this paper we present Peermarks, an application we developed to support social bookmarking, with the main purpose to study suitability of the peer-to-peer data management technologies we are researching. Peermarks allow users to share bookmarks through an XML and RDF-based, fully distributed datastore. The application allows users to classify web resources bottom-up (using tags) or to use a hierarchical categorization scheme. Peermarks supports both XML data storage and RDF-based inferencing. In Peermarks, users share bookmarks along with their classification system. We suggest the proposed system allows a global classification system to emerge from individual user categories.

\section{Background: Social Bookmarking and Folksonomies}

\subsection{Concepts}
Social bookmarking involves saving bookmarks to an online service and “tagging” them with keywords the user creates instead of saving the bookmarks in a browser’s favorites list. The collection of bookmarks can be made accessible to other users who may copy bookmarks to their own collection. Social bookmarking enables to discover other people who are interested in a topic and know about excellent web resources that one may not have found by using a search engine.

Tags can be thought of as keywords that allow ad hoc classification and sorting of a variety of types of information. Tagging, in the context of social bookmarking, is applied to URLs. Other popular web 2.0 sites (e.g. Flickr) apply the tagging approach to photos. A well understood problem with tag-based approaches to categorization of bookmarks (in general, of content) is that no consistent oversight exists to how
a resource is tagged—e.g. the tag London could stand for City of London or Jack London. A tag is just a tag—not a concept.

Traditional means of organizing information elements have generally relied on well-defined and pre-declared schemas ranging from simple controlled vocabularies to taxonomies to thesauri to full-blown ontologies. A controlled approach to cataloguing allows for both the validation and quality control of known terms to be registered within an information system. By contrast, tag-based systems create dynamic categorization systems whereby the user annotates links with whatever terms seem most relevant. Links are generally annotated with 'tags', which are free-form labels assigned by the user and not drawn from any controlled vocabulary. This is very much a 'bottom-up' (or personal) approach compared with the traditional 'top-down' (or organizational) structured means of classification [8]. This unstructured (or better, freely structured) approach to classification with users assigning their own labels is variously referred to as a 'folksonomy' or 'social classification', which is distributed and cooperative in nature.

Tags generally produce a flat namespace, rather than the hierarchical structures that a taxonomy or other formal classification system usually provides. This, of course, has its upsides and downsides. There is considerable debate in the specialized literature on the pros and cons of flat vs. hierarchical classification systems. Attempts are ongoing to introduce structure within tags. For example, tagging of tags (see for an example ‘tag bundles’ in del.icio.us) could help create hierarchical folksonomies. [4] suggests various possibilities to improve tag-based systems:

– ‘suggest tags for me’
– find synonyms automatically
– help me use the same tags others use
– infer hierarchy from the tags
– make it easy to adjust tags on old content

Currently only the last option appears to be in common use, presumably because it is the easiest to implement.

The application we developed, although our main aim was technological i.e. to test suitability of distributed query and inferencing techniques to a novel application fields, proposes also a fresh approach, which we believe a) helps users organize tags in groups, b) through sharing and collaboration, promotes cooperative, informal creation of taxonomies.

2.2 Related work

While social bookmarking is indeed a peer-to-peer practice, from a technology viewpoint most implementations we are aware of are centralized and do not use P2P technology.

Our review on related work hence focusses on P2P XML and RDF storage and query techniques.

Storing and querying machine-processable data is at the heart of the Semantic Web. However, the full potential is only unleashed with sophisticated reasoning algorithms, that allow to derive implicit knowledge from the explicitly encoded data. A problem
is the anticipated amount of data that renders computationally demanding reasoning algorithms unusable.

Thus, different streams of research can be seen in the Semantic Web community. On the one hand, the traditional description logics research focuses on tableau based reasoning that yields powerful reasoning features. However, tableau reasoners normally suffer from performance problems. Along many others, Pellet [15] is an example for a sophisticated tableau-based DL reasoner including state-of-the-art optimizations. Although Pellet is quite fast, its query answering speed cannot compete with relational databases.

On the other side of the spectrum, Sesame [5] is a fast and efficient RDF triple store that allows for RDF Schema semantics. It does reasoning by forward-chaining, effectively generating every possible conclusion. However this approach is limited in terms of the reasoning semantics.

The DL community has realized the need for semantically rich but yet fast reasoning mechanisms. Two prominent examples of this quest are EL [2] and DL-Lite [7]. EL is an simple DL dialect that has a polynomial reasoning complexity. DL-Lite is also a simplified DL that is executed using query rewriring. DL-Lite was the basic inspiration for our work, however we focus on a different set of basic constructs in the DL.

KAON2 [10] is a reasoner and Semantic Web framework that supports an expressive DL dialect. The DL reasoning is also done without a tableau algorithm. Instead, a translation to disjunctive datalog programs is employed.

The Jena framework [16] from HP is quite general, including various different reasoners. These reasoners encompass different trade-offs between semantics and performance.

In order to improve scalability several projects use distributed hash table P2P networks (DHTs) to store RDF data. Among these are RDFPeers [6], Atlas [11], RDF-Cube [13] and GridVine [1]. While the projects share common objectives they differ in their query processing and load-balancing strategies and capabilities. Edutella [14] follows a different route because it uses super-peer P2P networks as an underlying architecture.

### 3 Peermarking: Collaborative Creation of Taxonomies

Peermarks is basically designed to support a user, who searches the Web, to

- organize bookmark information about interesting Web resources for later recall
- discover potentially useful resources already ‘filtered’ by other users in the social network

In order to organize and classify bookmarks and to support later search and recall, a social bookmarking applications must use some categorization technique. In Internet-scale environments, where the classification is forcibly open or very large, it is very difficult or unfeasible to use an approach to categorization of resources based on controlled vocabularies.

Peermarks takes a mixed approach to categorization of resources for later retrieval.

- Web Resources can be bookmarked and associated to search terms in a bottom-up fashion by associating them to tags.
– In addition, the user can associate a web resource to a category from one or several classifications. Informally speaking, a category represents a concept, while a tag is just a name, possibly one of the several which can be related to the concept. Categories are arranged in a tree-like fashion, and retrieval works as one would intuitively expect - when querying resouring matching a category (e.g., winter sports), all resources associated to the category and its descendants in the tree are returned.
– Last, tags can be related to similar tags by relating them to a category.

As most social bookmarking tools, users can share their bookmarks and import bookmarks from other participants in the application. Peermarks makes a step further, by supporting users in collaborative creation of taxonomical and classification knowledge. This is achieved by allowing users to export their personal classification knowledge, which is merged by Peermarks into the network-wide, global classification knowledge. Users can then query the network classification and move selected parts of the network classification into their personal classification. See the section on Peermarks feedback loop, below, for a full description.

3.1 The Feedback loop

This section describes a typical workflow for a Peermaker user (see Figure 1).

Fig. 1. Peermarks: Workflow
1. **Bookmarking:** Users browse the web. As they find an interesting page, they can bookmark it with Peermarks. A tool to be installed into the browser toolbar is provided to make this operation easier. As the users bookmark a page, they can associate it with one or more tags.

2. **Working with categories and tags:** Users can browse through the bookmarked resources and organize them by associating each resource with one or more categories. Tags can also be associated to a concept/category, or a concept can be created from the tag.

![A bookmarked resource](image)

**Fig. 2.** A bookmarked resource

3. **Querying:** Resources can be queried both through filtering based on tags or navigating through the classification hierarchy.

4. **Exporting:** At any point in time, users can select a subtree of their personal knowledge base, and export it to the network (either the classification knowledge only or both the classification and the bookmarked resources).

5. **Importing:** Similarly, users can import a selected subset of the network classification knowledge and resources bookmarked by other users in the network into their personal workspace. This is stored at this stage into a logically separate information space (termed ‘network knowledge base’) from the personal knowledge base. Note: from a technical point of view, the network knowledge base is a sort of cache on the local machine of the network knowledge stored in the P2P datastore. Since Peermarks can be installed either as a single-user tool on e.g. a user laptop, or as a multi-user tool on a local server, the network knowledge base is shared by all the users of the local machine. See Architecture section.
6. **Browsing the network knowledge base**: In addition to querying the personal knowledge base, the user can search or browse the taxonomy and bookmarks imported from the network (see figure 3).

![Fig. 3. Subcategories of 'Programming' for personal and network view](image)

7. **Promoting network knowledge into the local view**: Finally, Peermarks allows users to see side-by-side information (e.g. descriptions, subcategories) for a category as seen a) by the user or b) by the network. This is achieved in the directory-tree view of the application, using tree widgets, showing the personal and network view, and in the category detail view when navigating the categories. As an example, in the local view Sports can have children “ski” and “mountain-bike”, while in the network view a more refined hierarchy exist accounting also for “winter sports”, “water sports” etc. (see figure 3). The user may decide to improve his local knowledge base by promoting those additional concepts into the categories she normally uses in her personal view (e.g. by dragging and dropping selected nodes between the two trees) (figure 4).

   We expect this feedback loop will support emergence and stabilization over time of a global classification, representing more accurately the common knowledge of the participants in the social network.

4 **Peer-To-Peer Storage and Retrieval of Share Bookmarks**

4.1 **High-Level Architecture**

Peermarks is implemented as a combination of a local, browser-based application with a peer-to-peer knowledge store. We already described the Peermarks from a logical viewpoint (see picture in previous section). A technical description follows now.
Peermark can be run either on a user’s personal workstation or laptop, or in a local server environment, serving multiple users (figure 5). Each Peermark installation provides a multi-user web application, supporting the workflow described in the previous section. Users access Peermark through a web browser and create, save and organize their resources on a persistent store, currently implemented as a MySQL DBMS. Additionally, each Peermark node runs a node of the BabelPeers peer-to-peer datastore, which is described below. The BabelPeers node collaborates with other nodes to implement the shared network datastore and allows the local node to access the network datastore. Network knowledge is cached (for all users in a local Peermarks installation) on the MySQL database and updated periodically or when a user requests a refresh.

4.2 Overview of BabelPeers

The main idea of BabelPeers is to provide a uniform view on knowledge that is distributed world-wide over a huge number of nodes. The system enables to query the network knowledge as a whole rather than asking nodes individually. Additionally, it is possible to also infer implicit knowledge from the network, especially knowledge that cannot be inferred when looking only at individual nodes.

To achieve this goal, BabelPeers uses the W3C standard Resource Description Framework (RDF) [12] for knowledge representation. RDF knowledge consists of triples of the form subject, predicate, object that can be regarded as sentences. Thus each triple is a single basic assertion of some knowledge.

A set of triples can be regarded as a so-called RDF graph. In BabelPeers, each connected node stores its local knowledge as an RDF graph. However, as RDF triples...
use URI’s for object reference, multiple RDF graphs that refer to the same objects can be merged into a single, large RDF graph. From this graph, new triples can be inferred using axioms from Ontologies encoded e.g. using RDF Schema or even the Web Ontology Language (OWL).

In large systems it is infeasible to collect and maintain the whole RDF graph at a single site. To cope with this scalability problem, BabelPeers uses instead a virtual graph that is in fact distributed over the whole network. For this, a distributed hash table (DHT) is used. Each RDF triple is stored three times on three different nodes, using its components as keys that determine the responsible nodes. Using this mechanism, it is possible to retrieve triples even if only a single component is known.

BabelPeers includes sophisticated query processing algorithms that allow the user to formulate complex queries using the W3C standard query language for RDF, named SPARQL. With this query language, graph patterns can be described that are searched efficiently in the distributed RDF graph. For details about the query algorithms, see [9].

Additionally, it is often desirable to infer new knowledge and to also return this inferred knowledge when answering queries. The overall result is shown in figure 6. Each node in the network has its own, local RDF graph. Through the DHT distribution, a large virtual graph spans over the network, including newly inferred edges (black edged in the picture). A query is not answered by looking at an individual node, but rather by exploiting the virtual distributed graph including the inferred edges.

BabelPeers supports two different mechanisms to infer new knowledge. For RDFS reasoning, a forward-chaining approach is used that generates every possible conclusion from the existing knowledge in advance. This maximizes query throughput, however the reasoning capabilities are limited. To overcome this problem, we have developed a different approach based on query rewriting.
The basic idea of this approach is that typically the conceptual part of the knowledge (TBox) is small compared to the instance data (ABox). Thus it is feasible to broadcast the TBox to every node in the network. The ABox is kept distributed as described above. As the whole TBox is now known locally at every node, it can be used to rewrite or reformulate the queries to reflect the axioms in the TBox. The result of the rewriting algorithm is typically a set of queries that can be evaluated using BabelPeers’ query evaluation strategies.

With this rewriting strategy, we are able to support a broad range of TBox axioms, including complex concept equalities, inverse roles, transitive roles, and domain and range restrictions for roles. The rewriting algorithm takes as input a conjunction of disjunctions, where each atomic element asserts the existence of some triple. Now the algorithm checks all axioms in the knowledge base to infer alternative patterns that have the same meaning. These alternative patterns are added disjunctively to the query. Thus the resulting query gets fairly large, however it delivers complete result w.r.t. the underlying semantics.

Further heuristics are used to reduce the size of the rewritten queries and to find a good execution order (query plan). For a complete description of the rewriting algorithm, see [3].

4.3 Merging Personal Classifications into a Global View

In Peermarks, BabelPeers is used to build the emerging global classification. As shown in figure 7, each Peermarks client first lets the user locally generate some classification, bookmarking, tagging, etc. This local information is synchronized with the BabelPeers peer. This peer stores it in its local RDF data store and starts distributing it from there over the network. Through this process, the global (but yet distributed) view emerges.

In the following, we describe the concepts and roles (i.e. the schema) used to store the classification. As the classification hierarchy is the user-supplied data, we store it
in the ABox rather than in the TBox. The TBox describes the used concepts and roles and contains general axioms. Using this approach the global classification is build automatically through our usage of the DHT to distribute the individual triples that make up the knowledge.

First, there is a concept Category that contains every user-defined category as an instance. For the category hierarchy, we use the hasChildCategory and hasParentCategory roles. These roles are defined as inverse roles, thus it is sufficient to either state the triple $Cat1 \text{ hasChildCategory } Cat2$ or the triple $Cat2 \text{ hasParentCategory } Cat1$. The existence of other triple will be inferred during query evaluation. Second, we use two roles named hasTransitiveChildCategory and hasTransitiveParentCategory, which are super-roles for hasChildCategory and hasParentCategory, respectively. As these roles are declared to be transitive in the Ontology, they can be used to query the transitive closure of the category hierarchy. So the application can query hasChildCategory to retrieve the direct children for a given category, and hasTransitiveChildCategory to get every descendent category.

Furthermore, we allow arbitrary relationships between the categories to declare a category to be related to another category. For this, a role hierarchy is used that is rooted at the role relatedCategory. This hierarchy is meant to be extended to more specifically describe the type of relationship between the categories. To classify tags or other resources, a role classified is used with range Category. This approach is very flexible, as anything can be regarded as a resource. It has just to be described through an URI. So bookmarks, tags, or anything else can be classified using the category hierarchy. RDF is very flexible in this respect, additional roles or concepts can be used to further describe the classified resources, making Peermarks open to further applications.

4.4 Querying the Global Classification

Querying the global classification is done using an additional API layer above BabelPeers that allows applications like Peermarks easy access and that hides the details
of the used Ontology and conversion of information into RDF triples. The API consists of functions like

```java
public Set<String> getChildCategories(String categoryName);
```

This function e.g. retrieves the direct child categories for a given category. It issues an SPARQL-Query via the BabelPeers query processor:

```sparql
SELECT ?child WHERE
{ cat:<categoryName> cat:hasChildCategory ?child . }
```

This query is straightforward, asking for every triple in the network that connects the selected category with a child category. However, due to the axioms in the underlying Ontology, `cat:hasChildCategory` is known to be an inverse role for `cat:hasParentCategory`. Thus, also triples matching the pattern `?child cat:hasParentCategory cat:<categoryName> .` are searched.

To retrieve the whole set of descendant categories from a given point, the following API call can be used:

```java
public Set<String> getAllChildCategories(String categoryName);
```

The implementation of this function is completely analogous to the above function, except that another role is used. Thus the SPARQL query is:

```sparql
SELECT ?child WHERE
{ cat:<categoryName> cat:hasTransitiveChildCategory ?child . }
```

This causes multiple rewriting rules to apply. First, the inverse role `hasTransitiveParentCategory` is also used. Second, as this role is a super-role for `hasChildCategory`, also this role is queried. Third, as `hasTransitiveParentCategory` is marked to be transitive in the Ontology, the transitive hull of the union of all results is built before applying the pattern.

This shows how relatively simple queries are reformulated to more complex queries that include reasoning logic. These more complex queries are evaluated in a distributed fashion by the BabelPeers network. BabelPeers includes various methods to minimize network traffic during the query evaluation, which assures efficient processing even in case of large networks with huge numbers of nodes and large classifications with many bookmarked resources.

### 5 Conclusion

While lab results exist for the peer-to-peer technology behind Peermarks, we developed the application to make it possible to deploy our peer-to-peer technology in real world usage. Although still a research prototype with several shortcomings we are aware of (security first of all have not been considered), we try to make Peermarks into an application which could be usable and appealing and could be tried by real users, at least in controlled environments. At this stage we have completed internal testing and we are releasing the application for free usage. We hope to gather practical insight from this deployment both on the performance and technical issues, as well as on the effectiveness of the Peermark feedback loop as an enabler to support emergence of shared classification knowledge.
References

URL citeseer.ist.psu.edu/aberer04gridvine.html