

Viewing a World of Annotations through AnnoVIP

Jesús Camacho-Rodríguez¹ Konstantinos Karanasos¹ Ioana Manolescu¹
Alin Tilea¹ Spyros Zoupanos^{1,2}

¹INRIA Saclay–Île-de-France, 4 rue Jacques Monod, 91893 Orsay Cedex, France ²U. Paris IX, France

firstname.lastname@inria.fr

Abstract

Le développement de contenus en format numériques a conduit à l'apparition de corpus de documents structurés interconnectés (tels que les pages HTML ou XML) et d'annotations sémantiques, typiquement exprimées en RDF, qui rajoutent des informations sur ces documents. Les annotations sont souvent produites indépendamment des documents. Nous présentons AnnoVIP, une plate-forme pair-à-pair capable d'exploiter de manière efficace un corpus de documents annotés, s'appuyant sur un nouveau modèle de vues matérialisées XML, déployées en pair-à-pair.

Keywords XML, RDF, annotated documents, materialized views, DHT.

1 Outline

In recent years, more and more software tools, including the most user-friendly ones, such as text editors, have started to export their contents into some *structured document format*, such as HTML or XML. Moreover, *annotations* have become very popular as a means to add information to a given document. HTML Meta tags, Dublin Core [17] and social networks' tagging are among the most common methods to express annotations. Here, we designate by annotation any simple statement in the style of the RDF standard [14], attaching to a given subject (or resource, such as a document, or a small portion of text) a named property, with a certain value.

Using documents *and* annotations provides the flexibility to handle a variety of application scenarios in which documents or RDF alone would not be suitable. As an example, think of a Web page containing a news item, annotated by a human reader or a text analysis tool to point out the person names appearing in the page, her positions within various institutions etc. or to express subjective opinions regarding the document. One could suggest modifying the Web page to incorporate the additional information of the annotations. However, this is not always feasible, since the author of the annotations may be distinct from, and have no control over, the author of the original document; furthermore, the original document should be readable also to those that are not interested in the extra information. Using only RDF to model all the content, on the other hand, is not appropriate since end users are familiar with, and expect to use structured documents.

Documents and annotations are at the center of the WebContent [18] project, in which we are involved. The project is focused on building and maintaining warehouses of enhanced Web documents on specific topics, e.g., market survey for the EADS european company (with offices in several countries) or an intelligence survey/warehouse concerning news from online media, bloggers etc.

Content publication in WebContent applications is inherently distributed. Web documents are fetched by crawlers at different sites, possibly re-formatted, translated from one language to another, and published by the respective sites. Similarly, documents and annotations authored by domain experts are published from their sites. WebContent applications require that all sites be able to exploit all the published contents. One could have considered uploading all published content to a single site. However, this raises some scalability problems and introduces a single point of failure.

To address such applications, we have built **AnnoVIP**, a symmetrical, peer-to-peer platform, based on a distributed hash table (or DHT, in short [5]), which guarantees upper bounds on the number of hops needed to route a given message in the peer network. At the core of content sharing in AnnoVIP stand *materialized views over the whole network content*. Each peer may locally store some XML and RDF content, and may also define views (stored locally as well), describing patterns of interconnected

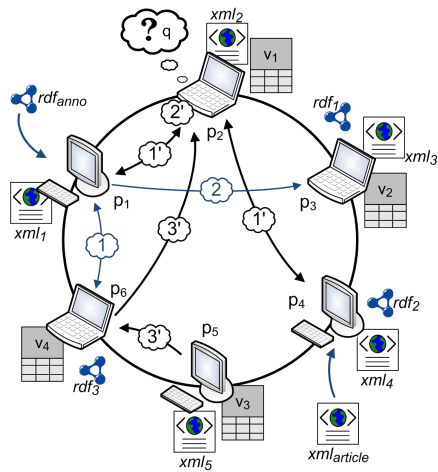


Figure 1: Architecture overview.

```
<article>
  <publishInfo>
    <author>Alice</author> <year>2009</year>
    <country>...</country> </publishInfo>
  <headline>Financial Crisis</headline>
  <topic>economy</topic> <body>...</body>
</article>
```

Figure 2: Sample document $xml_{article}$.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999-rdf-syntax-ns#"
  xmlns:anno="http://gemo.inria.fr/annotate/">
  <rdf:Description rdf:about="http://gemo.inria.fr/article.xml#body">
    <anno:authName>Bob</anno:authName>
    <anno:authCountry>France</anno:authCountry>
    <anno:date>26-6-2009</anno:date>
    <anno:rating>interesting</anno:rating>
    <anno:seeAlso>"http://gemo.inria.fr/article2.xml"</anno:seeAlso>
  </rdf:Description>
</rdf:RDF>
```

Figure 3: Sample annotation.

documents and annotations the peer is interested in. Once a view is established, its definition is indexed in the DHT. When documents or annotations are published, by looking up in the DHT, the publishing peer learns if its new content may contribute to some view, and if so, it sends the respective data to the view. After publication, this lookup is repeated periodically to identify contributions to later defined views. Thus, views are updated over time, in the manner of long-running, de-centralized subscriptions.

A further step in content sharing in AnnoVIP is *materialized view-based query rewriting*. Here, we consider the situation when a peer issues an ad-hoc query, which it has not declared as a local view. The peer then looks up in the DHT the existing view definitions, and may rewrite its query based on the views. From a rewriting, a distributed query plan is derived and evaluated. When available, pre-computed materialized views may lead to efficient query evaluation.

The novelty of AnnoVIP stems from its built-in dual support for documents and annotations at arbitrary granularity (one can annotate a document, an element, or even a text fragment). Maintaining and exploiting materialized views for efficient query processing over such interconnected corpora of documents and annotations requires new algorithms, further complicated by the distributed setting.

In the sequel, Section 2 discusses content publishing in AnnoVIP, whereas Section 3 describes querying. We outline the target demo scenarios in Section 4 and present related works in Section 5.

2 Content publication in AnnoVIP

In this section, we discuss publication of documents, annotations and views in our system. Section 2.1 describes the contents which one may publish, whereas Section 2.2 outlines the publication process. To illustrate our explanation, a simple AnnoVIP instance over six peers is depicted in Fig. 1. Next to each peer, the Figure shows its published XML documents (such as xml_1 , xml_2 etc.), annotations (denoted rdf_1 , rdf_2 etc.) and/or materialized views (v_1 , v_2 etc.).

2.1 Content model

The first kind of content we consider consists of XML documents. Each document d published by peer p has an URI allowing to uniquely determine d inside p and in the whole network. For example, Fig. 2 shows an article on the financial crisis, published by user Alice. This could be the $xml_{article}$ document published by peer p_4 in Fig. 1.

Annotations can target content at very different granularity levels. Thus, one can annotate a document, an element, a text node, or even a fragment of text, e.g., a significant phrase, or a person's name inside a text. Therefore, we consider that any fragment of a document d , whatever its size, has an URI. Moreover, the URI of d can be easily obtained from the URI of any fragment of d . This holds in many common URI schemes, such as XPointer [16] where $d.URI$ is a prefix of all the URIs of elements in d .

We assume any XML element has a child labeled URI , whose value is its actual URI. However, URI-labeled nodes are virtual, as they do not actually appear in elements (although as we will explain, they are needed for querying purposes). In a similar manner, any fragment can be seen as a node, endowed with an URI. Without loss of generality and for conciseness, we focus on elements hereafter.

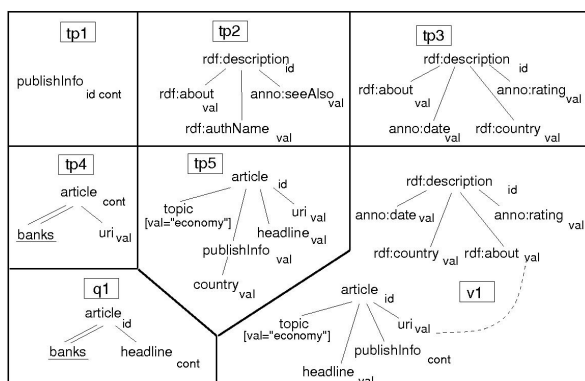


Figure 4: Sample queries and views.

The second class of content we consider concerns annotations, which may be either produced by human users, possibly with the help of some tools, or by automated modules (e.g. recognizing named entities within a document). For simplicity, we consider that all annotations have been brought to an RDF format. Thus, the basic unit of content here is a triple $t=(s, p, o)$, specifying the value o of property p for the resource s . We assume triples are serialized following the XML syntax for RDF [15].

For instance, let us consider how user Bob at peer p_1 in Fig. 1 produces the new annotation rdf_{anno} . Assume Bob discovers the $xml_{article}$ published by Alice at p_4 and decides to annotate the *body* of the article as being “interesting”. Bob also suggests another article ($article2.xml$), which he finds related. The corresponding RDF annotation appears in Fig. 3, together with the author’s (i.e., Bob’s) profile.

2.2 Peer-to-peer views in AnnoVIP

Views and queries are defined in the same language, which can be seen as joins over a specific flavor of tree patterns. Fig. 4 shows some examples.

Each tree pattern node carries a name label, corresponding to an element or attribute name or a word appearing in a text node in some document or annotation (we use \underline{w} to denote the word w). Pattern edges correspond either to child ($/$) or descendant ($//$) relationships between nodes; we assume that a text word is a child of its closest enclosing element. *Due to the special role we attach to URIs, we impose that an URI-labeled node always appears as a child (not descendant) of another node in the view.* Each node may be decorated with zero or more among the following labels: *id*, standing for *structural identifier*¹; *cont*, standing for the full XML subtree rooted at the node; and *val*, standing for the concatenation of all text descendants of the node. An *id*, *cont* or *val* label attached to a node denotes the fact that the structural ID, the content or the value, respectively, of the node belong to the pattern result. Finally, each node may be labeled with a predicate of the form $[val=c]$ where c is some constant.

In Fig. 4, pattern tp_1 stores the structural ID and the content of all *publishInfo* elements. Pattern tp_2 stores the author’s name, as well as the suggested documents for all annotations, while tp_3 keeps the date, country and rating of annotations. Pattern tp_4 stores the URIs of all articles containing the word “banks”. Observe that although element URIs are not stored in the original document, they must be actually stored in a view such as tp_4 . Finally, tp_5 stores the URI, the headline and the country of the publisher of all articles about the economy.

More complex views can be obtained by joining tree patterns based on some value equality predicates. For instance, a view may require the name of each author having annotated an article which mentions “banks”, by connecting tp_2 and tp_4 using a value join ($rdf:about.val=uri.val$). Observe that the presence of value joins in the language is crucial for capturing the connections between content and annotations on that content, via such equality predicates.

Let us consider how published views are filled with data. In the simple case of tree pattern views, the tree pattern defining the view is indexed in the DHT by the labels of all its nodes. Assume peer p publishes some new content. Then, p will perform lookups on all the element names and words appearing in this content, to find (possibly a superset of) views to which the new content may contribute. p then evaluates these views against its content. The evaluation leads to a set of tuples, which p sends

¹By comparing the structural identifiers of nodes n_1 and n_2 , one can decide whether n_1 is an ancestor of n_2 or not. Many popular examples exist, e.g. [4, 12].

to the peer which published v . Thus, the extent (tuples) of a view accumulates, and is stored, at the view publishing peer. Periodic view lookup is performed to capture the case when documents and annotations are published after a view to which they may contribute.

For example, Fig. 1 shows the events taking place when user Bob publishes $rd\dot{f}_{anno}$ at p_1 . First, he performs a look up to determine the view definitions to which the new annotation may add some tuples (step 1). In our Figure, p_6 holds a view definition index entry referring to such a view. Upon receiving the view definition, say that of v_2 which p_3 holds, p_1 extracts the tuples corresponding to $v_2(rd\dot{f}_{anno})$ and sends them to p_3 (step 2), which appends them to the view extent. Observe that the contribution of $rd\dot{f}_{anno}$ to the view could be evaluated locally at p_1 , assuming the view is a tree pattern.

This resource publication process needs to be extended to accommodate views with joins. Indeed, in such cases, a peer publishing some new content needs to know if and where, in the whole network, some other content may satisfy a value join with its own. One solution is to maintain, instead of a join pattern view, one view per each tree pattern, and compute view tuples incrementally as new tuples are added to each tree pattern, in the style of incremental maintenance for join views [7]. However, this may lead to accumulating an unbound amount of data, if, for instance, many documents matching one view tree pattern are published, which do not join with any other tree pattern. To deal with this shortcoming, we devised more efficient techniques both for the cases when join predicates do and do not involve *URI attributes* [8]. We notice that joins over URI attributes are present whenever a view queries documents with annotations.

3 Query rewriting using views

We now consider the processing of a query, such as q in Fig. 1, posed at a peer p_2 which has not declared q as a local materialized view. First, p needs to find out which views in the network could possibly be used to answer q . This relies on the same view definition indexing used for view maintenance. Then, p runs a view-based rewriting algorithm to find complete rewritings of q using the existing materialized views. Finally, one rewriting is picked and evaluated over the distributed peers.

For instance, in Fig. 1, query q is posed by user Carole at peer p_2 . To find view definitions relevant to q , p_2 performs a DHT look-up (step 1'). Assume that peers p_1 and p_4 return relevant view definitions. Then, p_2 tries to rewrite q based on these views (step 2') and gets a logical algebraic plan based on some views, in our example v_3 and v_4 , which are stored in peers p_5 and p_6 respectively. Subsequently, p_2 transforms the rewriting to a physical plan, which is executed in a distributed manner (step 3'). For instance, v_3 is sent from p_5 to p_6 , where it is joined with the local v_4 and the result is sent back to p_2 .

AnnoVIP's rewriting algorithm restricted to tree pattern queries is provided in [10], and extended to the general language in [8]; for space reasons, we do not detail it further here. We illustrate it via examples (recall again the sample queries in Fig. 4).

Query q_1 asks for the headlines of articles mentioning banks. A possible rewriting could use tp_4 to navigate through the content of element *article* and then project the headline. Now consider a more complex query q_2 , demanding the headlines of articles published in 2009, annotated as interesting today in France. Query q_2 can be rewritten using tp_3 and tp_5 in the obvious way, if these patterns are available as materialized views. The drawback is that this rewriting still requires evaluating a join. However, if view v_1 was available, q_2 could be answered more efficiently, without evaluating any costly joins.

4 Platform and Demonstration Scenario

We have implemented AnnoVIP on top of VIP2P [10], a peer-to-peer platform for sharing XML documents using materialized views. The extension we brought concerns the support for annotations, value joins in the query language, and specific techniques for materializing join views and rewriting queries including value joins. The system is fully implemented in Java, using the Pastry DHT and BerkeleyDB for storing materialized views. Screenshots and other information about it can be found at <http://vip2p.saclay.inria.fr>. VIP2P's scalability has been established in large-scale experiments involving up to 1000 peers in a country-wide WAN [1], thousands of documents and hundreds of views.

During the demonstration, attendees will get a hands-on experience on deploying and running AnnoVIP network. We plan to reserve a few hundreds of machines in the Grid5000 network [1], on which will be deployed a set of documents, views, and annotations to start with. For demonstration purposes, one peer will play the role of the coordinator, to which all others report their events (publishing views, documents or annotations, adding data to views etc.). Demo attendees will then get a chance to provide their own content in the network either from sample files which we will provide, or by importing various Web pages; we will also use the RSS stream of a few blogs. At that point, annotations at various granularities could be added on top of the documents. Then, the content of any view could be dynamically inspected via the GUI to see how it changes as new contributions are added to it. A visual representation of the network topology will also be available, enabling demo attendees to trace the distributed resource publication or query processing.

5 Related works

Our demonstration relates to many works on XML indexing in DHT networks [3, 6, 11], over which it improves by allowing to declare and exploit complex materialized views to speed-up the processing of specific application queries. In [10], we have provided algorithms which handle efficiently tree pattern views in large networks (1000 peers). AnnoVIP is the first to jointly consider documents and annotations, and as a consequence, as explained in Section 2, we add value joins to the views and to the queries. Numerous recent works have targeted efficient RDF querying in a centralized setting [13] and based on DHTs [9]. The specificity of our work is to first, combine documents and annotations and second, focus on view maintenance and view-based rewriting.

Within the WebContent project [18], we have devised a DHT-based platform integrating two types of DHT content indices [2]. However, this still did not provide sufficient leeway to establish efficient data access support structures. Moreover, the biggest performance problems we encountered while using the system [2] were due to the frequent joins generated by document-and-annotations queries.

References

- [1] Grid'5000 network infrastructure. <https://www.grid5000.fr/>.
- [2] S. Abiteboul, T. Allard, P. Chatalic, G. Gardarin, A. Ghitescu, F. Goasdoué, I. Manolescu, B. Nguyen, M. Ouazara, A. Somani, N. Travers, G. Vasile, and S. Zoupanos. Webcontent: efficient P2P warehousing of web data (demo). *PVLDB*, 1(2), 2008.
- [3] S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun. XML processing in DHT networks. In *ICDE*, 2008.
- [4] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural joins: A primitive for efficient XML query pattern matching. In *ICDE*, 2002.
- [5] F. Dabek, B. Zhao, P. Druschel, J. Kubiawicz, and I. Stoica. Towards a common API for structured P2P overlays. In *IPTPS*, 2003.
- [6] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Locating data sources in large distributed systems. In *VLDB*, 2003.
- [7] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *SIGMOD Conference*, 1993.
- [8] K. Karanasos and I. Manolescu. P2P Views over Annotated Documents. <http://vip2p.saclay.inria.fr/papers/report.pdf>, 2009. Technical report.
- [9] E. Liarou, S. Idreos, and M. Koubarakis. Evaluating conjunctive triple pattern queries over large structured overlay networks. In *ISWC*, 2006.
- [10] I. Manolescu and S. Zoupanos. Materialized views for P2P XML warehousing. <http://vip2p.saclay.inria.fr/papers/paper.pdf>. Submitted for publication.
- [11] P. Rao and B. Moon. An internet-scale service for publishing and locating XML documents (demo). In *ICDE*, 2009.
- [12] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *SIGMOD Conference*, 2002.
- [13] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1), 2008.
- [14] RDF: Concepts and Abstract Syntax. <http://www.w3.org/TR/rdf-concepts/>, 2004.
- [15] RDF/XML Syntax Specification. <http://www.w3.org/TR/rdf-syntax-grammar/>, 2004.
- [16] XML Pointer Language. <http://www.w3.org/TR/WD-xptr>, 2001.
- [17] Dublic core metadata initiative. <http://www.dublincore.org/>.
- [18] The WebContent Semantic Web platform. www.webcontent.fr.