Growing Triples on Trees: an XML-RDF Hybrid Model for Annotated Documents

François Goasdoué[1] Konstantinos Karanasos[1] Yannis Katsis[2]
Julien Leblay[1] Ioana Manolescu[1] Stamatis Zampetakis[1,3]

[1]Leo team, INRIA Saclay and LRI, Univ. Paris-Sud
[2]INRIA Saclay, ENS Cachan [3]Univ. of Crete
firstname.lastname@inria.fr

ABSTRACT

Content on today's Web is typically document-structured and richly connected; XML is by now widely adopted to represent Web data. Moreover, the vision of a computer-understandable Web relies on Web (and real world) resources described by simple properties having names or values; URIs are the normative method of identifying resources and RDF (the Resource Description Framework) enjoys important traction as a way to encode such statements.

We present XR, a carefully designed hybrid model between XML and RDF, for describing RDF-annotated XML documents. XR follows and combines the W3C's XML, URI and RDF standards by assigning URIs to all XML nodes and enabling these URIs to appear in RDF statements. The XR management platform thus provides the capabilities to create and handle interconnected XML and RDF content. We define the XR data model, its query language, and present preliminary results with a prototype implementation.

1. INTRODUCTION

With its widespread acceptance, XML has become the language of choice for publishing semi-structured data on the Web, be it business, scientific or governmental data.

Efficient and effective search for information within large XML corpora is thus crucial. Optimizing the XML storage and querying infrastructure is a well-trodden path towards increasing efficiency. However, on heterogeneous Web data authored in a distributed, independent fashion, one could greatly improve search effectiveness by querying XML sources jointly with *data about the documents*. In particular, if semantic information about the XML documents is available, not only additional information can be exploited during the search process, but also new knowledge can be discovered through a reasoning process.

Consider a newspaper publishing its articles in XML. To enhance the way readers search for information, the administrator adds annotations to the articles, classifying them according to their topics selected from a publicly available ontology on news articles. By reasoning on the ontology, a user searching for article topics (e.g., biology), also obtains articles on more specific ones (e.g., bioengineering, biofuel energy, etc.). The readers can also annotate the documents stating their personal opinion on an article or on specific parts of it. Moreover, articles can be linked with each other. For instance, when a specific article is also published in a blog, this version can be linked to the newspaper version. Hence, when searching for articles, it becomes feasible to use semantic information that is not attached directly to a given article, but to another linked version of it.

Many more use cases of semantic annotations can be found. Among them, and closely related to the Web search, lately, search engines such as Google Search and Microsoft Bing, let users reward search results they like or use information available through social networks (e.g., friends list) to return results more suitable for the users.

Such annotations can be naturally expressed in RDF, which is becoming the de facto standard for describing semantically rich data. First, by assigning URIs to any part of the XML articles, we can refer to these parts in RDF statements and, thus, add semantic information on the articles at any granularity, all the while leaving the original documents intact. Second, in the spirit of the Linked Data initiative (http://linkeddata.org), we can create semantic links between documents. Finally, RDF, especially when combined with RDF Schema (RDFS in short), enables the discovery of new implicit knowledge through reasoning.

Previous works have enabled the simultaneous processing of XML and RDF data, typically by converting RDF to XML (or vice-versa) and then using a language and platform dedicated to the target format. Such translation results are generally awkward and bury content under syntax. Fundamentally, the data models are quite different: XML emphasizes structural relationships, whereas RDF consists of triples connected in arbitrarily complex graphs. As a consequence, the challenges and opportunities for optimization are very different for the two models. Thus, existing optimizers and techniques for one model are likely not to apply well on the data converted from the other model.

At a more conceptual level, these approaches *juxtapose* XML and RDF data, and do not consider the main idea of this work, namely, turning XML nodes (or words) into RDF resources, and *connecting* these data sources.

In this work, we propose a unified model allowing the combination of XML with RDF data into a single instance. Our main contributions are: (*i*) a data model for annotated documents which can express instances where XML and RDF are interconnected (described in Section 2), (*ii*) a query language for querying the unified data model (Section 3) and (*iii*) an implemented system with experiments encompassing the previous ideas (Section 4). The full technical report describing our work is available at [8].

2. THE XR DATA MODEL

To represent annotated documents, we introduce the XR data model, extending and combining the standard XML structured data model and RDF semantic data model. An instance of the XR data model comprises two *sub-instances*: An XML sub-instance, consisting of a set of XML trees, and an RDF one, consisting of a set of RDF triples. The connection between the two is achieved by assigning to each XML node a unique URI, which can then be referred to from an RDF triple, as we will explain below.

The XML sub-instance relies on a set $\mathcal{N}$ of possible XML element and attribute names, a set $\mathcal{U}$ of URIs, and a set $\mathcal{L}$ of literals. An XML tree is defined as usual:

DEFINITION 2.1 (XML TREE). *An* XML tree *is a finite, unranked, ordered, labeled tree* $T = (N, E)$ *with nodes* N *and edges* E*, where each node* $n \in N$ *is assigned a label* $\lambda(n) \in \mathcal{N}$ *and a type* $\tau(n) \in \{attribute, element, text\}$*. An attribute node must be the child of an element node; it has a value belonging to* $\mathcal{L}$ *and it does not have any children. A text node can only appear as a leaf.*

A set of XML trees forms an XML instance:

DEFINITION 2.2 (XML INSTANCE). *An* XML instance I_X *is a finite set of XML trees together with a function assigning to each node of these trees a unique URI from* $\mathcal{U}$*.*

The URI assignment function is crucial for interconnecting the XML and RDF sub-instances. The unique identifiers assigned to the nodes allow the RDF sub-instance to refer to nodes of the XML sub-instance.

In addition to the aforementioned sets $\mathcal{U}$ and $\mathcal{L}$, the RDF sub-instance also relies on a set of blank nodes $\mathcal{B}$, whose role is discussed below.

DEFINITION 2.3 (RDF INSTANCE). *An* RDF instance I_R *is a set of triples of the form* (s, p, o)*, where* $s \in (\mathcal{U} \cup \mathcal{B})$*,* $p \in \mathcal{U}$*, and* $o \in (\mathcal{L} \cup \mathcal{U} \cup \mathcal{B})$*.*

As customary, the components of a triple (s, p, o) are referred to as its *subject*, *property* and *object*, respectively.

As defined above, the subject or the object of a triple can be bound to a *blank node*. Blank nodes are used in RDF [13] to denote *unknown URIs* or *literals*, similarly to *labeled nulls* in the database literature [1]. For instance, one can use a blank node b_1 in the triples $(b_1, country, \text{“}France\text{”})$ and $(b_1, city, \text{“}Paris\text{”})$ to state that the *country* and *city* of b_1 is *France* and *Paris*, resp., without using a concrete URI.

Another advantage of adopting RDF is that XR inherits its reasoning capabilities. As described in [13], an RDF data set contains not only explicit triples but also implicit triples, derived from the explicit ones through a set of *entailment rules*, the most important of which are due to knowledge encoded in RDF Schemas. In the example discussed in the introduction, an RDF Schema specifies that bioengineering is a *subClassOf* biology. Assuming the paragraph p_1 is related to bioengineering, modeled by the explicit RDF triple

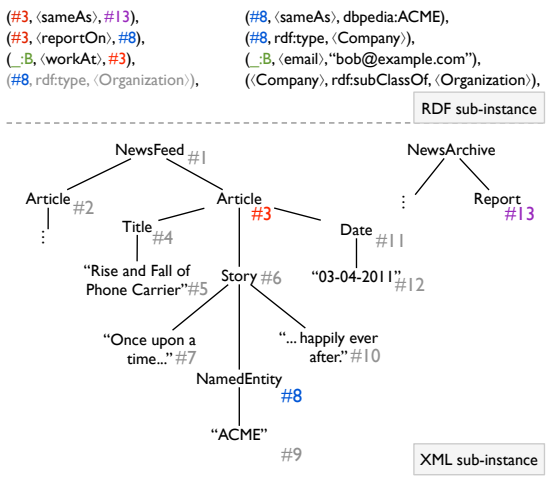

Figure 1: XR instance featuring an annotated news article.

(p_1, relatedTo, bioengineering), the implicit triple (p_1, relatedTo, biology) is also conceptually part of the data set. Implicit tuples must be reflected in RDF query answers. Accordingly, XR queries (presented in Section 3) also consider that implicit triples are part of the RDF sub-instance. The management of implicit RDF (and thus, XR) triples is an orthogonal issue, on which we will not delve further here.

We can now define an XR instance as follows:

DEFINITION 2.4 (XR INSTANCE). *An* XR instance *is a pair* (I_X, I_R)*, where* I_X *and* I_R *are an XML and an RDF data instance, respectively, built upon the same set of URIs.*

Note that the XML and the RDF sub-instances are defined over the same set of URIs $\mathcal{U}$, thus allowing RDF triples to annotate XML nodes, as shown in the following example.

Example. Figure 1 shows an XR instance. The XML sub-instance (in the bottom of the Figure) consists of two trees. The first represents a news feed composed of articles. Each article has a story (consisting of text nodes and a named-entity identifying a company name), a title and a date. The second tree refers to a news archive. Each XML node has a subscript corresponding to its URI. The RDF sub-instance comprises RDF triples which refer to (annotate) the XML sub-instance by using the URIs of the XML nodes. For instance, we state that the article with URI='#3' is a report on the named-entity with URI='#8', which is of type *Organization*. Moreover, in the context of Open Linked Data, we use the *sameAs* property to show that two resources are the same, e.g., the nodes with URIs '#3' and '#13' in the Figure. RDF triples may also contain blank nodes (e.g., $\_:B$). Finally, some triples may be implicitly derived, such as the one stating that the named-entity is an *Organization*, which was inferred from the facts that the *Company* is a subclass of *Organization* and the named-entity is a *Company*.

3. THE XRQ QUERY LANGUAGE

Given an XR data instance, users should be able to query the data based on both its structure, described in the XML sub-instance, and its semantic annotations, stored in the RDF sub-instance. To this end, we design the XRQ query language. In XRQ, the XML sub-instance is queried using *tree patterns*, while the RDF sub-instance is queried through *triple patterns*. Both are defined below. Most importantly, reusing variables across tree patterns and triple patterns allows to relate nodes and resources of the two sub-instances.

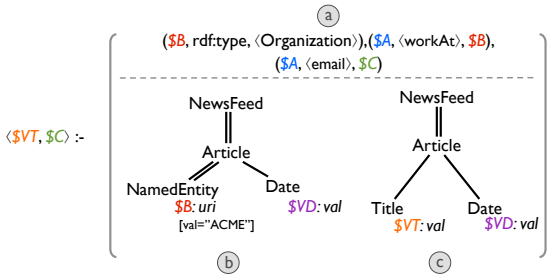

Figure 2: Sample XRQ query.

Definition 3.1 (Tree Pattern). *A tree pattern is a finite, ordered, unranked, $\mathcal{N}$-labeled tree with two types of edges, namely child and descendant edges. We may attach to each node at most one* uri *variable, one* val *variable and one* cont *variable. We may also attach to a node an equality predicate of the form* [val=c] *for some $c \in \mathcal{L}$.*

A tree pattern is a variant of tree patterns as presented in the literature [3] with the additional capability of attaching one or more variables of different types to the nodes. Variables serve two purposes: (i) to denote data items that are returned by the query (in the style of distinguished variables in conjunctive queries) and (ii) to express joins between tree or triple patterns. The variable type specifies the exact information item from an XML node, to which the variable will be bound. When a node n_t of a tree pattern is matched against a node n_d of an XML tree, the variables attached to n_t will be bound to the following concepts. A *uri* variable is bound to the URI of n_d. If n_d is an element, a *val* variable is bound to the concatenation of all text descendants of n_d; if n_d is an attribute, a *val* variable is bound to the attribute value. Finally, a *cont* variable is bound to the serialization of the subtree rooted at n_d. The semantics of *val* variables follow from the XPath and XQuery specifications [16].

Example. The lower part of Figure 2 depicts two tree patterns. Pattern (b) finds articles containing a named-entity with a value equal to "ACME". Variables $\$B$ and $\$VD$ are bound to the URI of this entity and the publication date of the article, respectively. Observe that $\$VD$ appears also in the second tree-pattern, thus expressing a join between them. Pattern (c) retrieves the titles of articles whose publication date matches that of pattern (b).

Definition 3.2 (Triple Pattern). *A triple pattern is a triple (s, p, o), where s, p are URIs or variables, whereas o is a URI, a literal, or a variable.*

Example. The top part of Figure 2 depicts three triple patterns, looking for entities of type *Organization* and finding emails of resources known to be working at that organization. Joins between triple patterns are expressed through shared variables, such as $\$A$ and $\$B$ in the Figure.

By combining tree and triple patterns and endowing them with a set of head variables, we obtain an XRQ query:

Definition 3.3 (XRQ Query). *An XRQ query consists of a* head *and a* body. *The body is a set of tree and/or triple patterns built over the same set of variables, whereas the head is a list of variables appearing also in the body.*

Note that by using variables in multiple places within the query, one can express joins. In general, three types of joins are possible: between two tree patterns, between two triple patterns or between a tree pattern and a triple pattern. The latter facilitates queries that cross the boundaries between XML documents and their RDF annotations. The following example illustrates the expressivity of XRQ.

Example. Figure 2 shows an XRQ query, whose body (at the right-hand side) consists of the previously described tree and triple patterns. Observe that variable $\$B$ appears in both parts of the body, forming a join between the XML and RDF sub-instances. Two variables appear in the head of the query: $\$VT$ from tree pattern (c) and $\$C$ from the first triple pattern. The answer set for this query will contain the titles of articles published on the same date as articles referring to the organization called ACME, and emails of people working in this organization.

Query Semantics For each match of the tree patterns, triple patterns and the corresponding variables against the XR instance (including the XML data, the explicit as well as the implicit triples, as explained in Section 2), we create a copy of the head tuple with the variables replaced by their bound values. The query answer is the set of all such tuples. For more details see the extended version [8].

In [8] we also provide an extended version of XRQ, supporting more complex construction clauses, such that the result of an XRQ query is an XR data instance.

4. EXPERIMENTS

To experiment with XR, we implemented XRP, a prototype platform that supports the storage of XR instances and the evaluation of XRQ queries, using Java 1.6.

Architecture The data store is built on top of BerkeleyDB. RDF triples are mapped to a simple three-attribute relation. XML documents are stored also within a tabular format, enriched with XML-specific features such as structural XML identifiers and full serialized XML fragments. For the experiments described here, we manually specified the XML data that each relation will store, by means of a tree pattern. XRP supports easily changing the data organization by specifying other tree patterns, thanks to its reliance on the ViP2P (http://vip2p.saclay.inria.fr) access path selection infrastructure [11]. Furthermore, we materialized an index of all *XML node URIs appearing in some RDF triples*. This index facilitates the evaluation of the XRQ queries that combine data from the two sub-instances.

The query engine supports the typical operators: scan, selection, projection, joins on values and XML structural IDs, etc. To take advantage of indexes, we also implemented the sideways information passing BindJoin operator [6], which uses attribute values from one join input as keys to access the other input (an XML/RDF index is used to retrieve for each XML node URI, all the triples containing this URI).

Experimental Setup For the purpose of the experiments, we created a synthetic XR instance with soccer league information. The XML sub-instance describes teams and players, while the RDF sub-instance annotates players with properties. To assess the system's scalability, we varied the number of player nodes (10K, 50K and 100K), while keeping the size of the XML document constant (95MB) by modulating the length of some attributes that each player contains. The number of annotations also increased proportionally by keeping an average of one annotation per player.

We devised three queries, each joining a tree pattern with a triple pattern on URIs of player nodes. Q_1 retrieves teams which have players annotated with a given property. Q_2 is a relaxed version of Q_1 returning teams regardless of the property used. Finally, Q_3 returns properties annotating

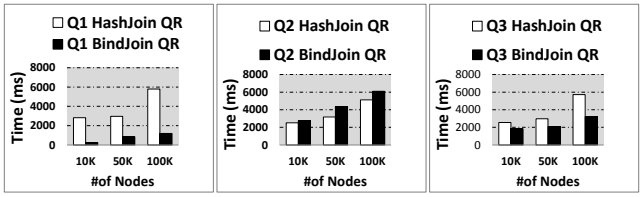

Figure 3: Query Response (QR) time for Q_1, Q_2, Q_3 on data sets of varying number of players (and triples).

players of a particular team. Observe that Q_1 and Q_3 are highly selective; Q_1 applies a selection on the RDF instance, whereas Q_3 applies a selection on the XML instance.

For each query, we compared execution times both with and without indexes on the joined attributes. For Q_1 and Q_2, we used an index on the XML node URIs, while for Q_3, we used an index on the RDF subjects.

The experiments were conducted on a 2.40GHz Intel X3430 with 4GB RAM running Mandriva Linux (kernel 2.6.31.14).

Experimental Results Figure 3 shows the total query evaluation times for each query over the three instances both with indexes (BindJoin) and without (HashJoin).

The results indicate that XRP scales well with the number of nodes in the XML sub-instance (and the number of triples in the RDF sub-instance). For instance, evaluating the least selective query (Q_2) over the largest data instance (100K) requires less than 6.5 seconds.

As expected, the BindJoin plans (using the index) are generally faster (up to a factor of 4 in some cases). The only case when BindJoin is slightly worse than HashJoin is for non-selective queries (such as Q_2). This is due to the overhead incurred from accessing all items of an index. Neither side of the join operator is more selective than the other and thus the join operator has to match all RDF triples with all tuples from the XML instance. BindJoin will need random access to the whole index, while HashJoin will simply apply a sequential scan to the same relation.

Our experiments demonstrate the applicability of many query and storage optimization techniques to the problem of XRP data management, in a single-platform setting. To better exploit existing systems, we are also investigating the deployment of an XR platform as an integrator, complementing separate XML, respectively, RDF databases, while preserving the single-model abstraction.

More experiments and a detailed description of the data sets can be found in our extended report [8].

5. RELATED WORK & CONCLUSION

We briefly discuss the main classes of related works.

Tools for annotating web pages Several works propose frameworks that let users semantically annotate web-pages either manually [9] or (semi-)automatically [4] (see [12] for a survey of such systems). However, they focus solely on storing and querying annotations and they do not consider the problem of simultaneously querying structured data and the annotations on them.

Embedding RDF annotations in XML documents Other works look at the particular problem of embedding RDF annotations in XHTML [14]. However, this requires modifying the original XML document, which is not always possible, due to privacy reasons or access constraints. In contrast, in XRP the RDF annotations can be kept separately from the XML documents in their own physical store.

From RDF to XML and back Another line of work studies the connection between RDF and XML. This includes works on (i) transforming XML data to RDF and vice versa [2], (ii) using the query language of one model to query the other (e.g., use XQuery to query XML-ized RDF) [5, 10] and (iii) extending the query language of one model with primitives for the other (e.g., adding XPath constructs to SPARQL)[15]. Finally, XML and RDF are modeled within a single rule-based framework in [7].

As outlined in Section 1, transforming XML to RDF or vice-versa is feasible but brings a significant conversion overhead, while the optimization techniques appropriate for one are likely not very efficient for the other. The main difference between our work and the previous ones integrating XML and RDF, though, is our treatment of any XML node as a resource, and the resulting focus on connecting (in the RDF/Semantic Web sense) data to and from documents.

Conclusion XRP is a first step towards enabling the modeling and querying of annotated XML documents. Our current and future work focuses on (i) checking the model's expressiveness against large-scale real-life annotated document applications built on previous content management systems such as Confolio (www.confolio.org), and (ii) devising an architecture for XR management based on off-the-shelf efficient XML and RDF data management platforms.

6. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds - and avoiding the XSLT pilgrimage. In *ESWC*, 2008.

[3] S. Amer-Yahia, S. Cho, L. V. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *SIGMOD*, 2001.

[4] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. SemTag and seeker: bootstrapping the semantic web via automated semantic annotation. In *WWW*, 2003.

[5] M. Droop, M. Flarer, J. Groppe, S. Groppe, V. Linnemann, J. Pinggera, F. Santner, M. Schier, F. Schöpf, H. Staffler, and S. Zugal. Bringing the XML and Semantic Web Worlds Closer: Transforming XML into RDF and Embedding XPath into SPARQL. In *ICEIS*. 2009.

[6] D. Florescu, A. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of binding patterns. In *SIGMOD*, 1999.

[7] T. Furche, F. Bry, and O. Bolzer. Marriages of Convenience: Triples and Graphs, RDF and XML in Web Querying. In *Principles and Practice of Semantic Web Reasoning*. Springer Berlin / Heidelberg, 2005.

[8] Extended version of this work, available at http://xr.saclay.inria.fr/papers/xr-vlds-extended.pdf, 2011.

[9] S. Handschuh and S. Staab. Authoring and annotation of web pages in CREAM. In *WWW*, 2002.

[10] K. Karanasos and S. Zoupanos. Viewing a world of annotations through AnnoVIP (demo). In *ICDE*, 2010.

[11] I. Manolescu, K. Karanasos, V. Vassalos, and S. Zoupanos. Efficient XQuery rewriting using multiple views. In *ICDE*, 2011.

[12] L. Reeve and H. Han. Survey of semantic annotation platforms. In *ACM SAC*, 2005.

[13] RDF. http://www.w3.org/standards/techs/rdf.

[14] RDFa. http://www.w3.org/TR/xhtml-rdfa-primer/, 2004.

[15] SPAT - SPARQL Annotations. http://www.w3.org/2007/01/SPAT/, 2007.

[16] XQuery 1.0 and XPath 2.0 Data Model (XDM). http://www.w3.org/TR/xpath-datamodel/, 2010.