Marriages of Convenience: Triples and Graphs, RDF and XML in Web Querying

Tim Furche, François Bry, and Oliver Bolzer

Institute for Informatics,University of Munich,
Oettingenstraße 67, 80538 München, Germany
http://pms.ifi.lmu.de/

Abstract. Metadata processing is recognized as a central challenge for database research in the next decade. Already, novel desktop data management and search applications (cf. Apple's Spotlight and Microsoft's WinFS) are enabled by rich metadata. Efficient and effective access to such data becomes a crucial issue for more and more application scenarios. In this article, we focus on metadata represented in RDF. A number of query languages for RDF have been presented in recent years. This article argues that most of these approaches fail to address properly two core issues: the provision of rich operators and constructs to adequately support RDF's graph data model and the ability to intertwine access to metadata (in RDF format) and data (in XML format). To address this points, two XML views over RDF data are expressed in the query language Xcerpt and discussed. Furthermore, it is shown how these views together with Xcerpt's rich graph patterns allow the succinct expression of complex, but common queries against RDF graphs.

1 Introduction

The 'Semantic Web' is an endeavor widely publicized in [1], envisioning the current Web, which consists of (X)HTML and documents in other XML formats, extended by metadata specifying the meaning of these documents in forms usable by both human beings and computers.

The integral processing of data and metadata is recognized as a central challenge for the next decade (cf., e.g., Pat Selinger's ICDE 2005 Keynote) not only as a contribution to the Semantic Web vision, but also on a smaller scale as part of the next generation of desktop data management (cf. Apple's Spotlight and Microsoft's WinFS that aim at extending current file storage and desktop search with extensive metadata facilities).

In the (Semantic) Web context, a number of formalisms have been proposed for representing metadata, in particular RDF, Topic Maps, and OWL. This article concentrates concentrate on RDF as the most widely used formalism. This article illustrates first steps towards integrating access to standard Web data in XML format and RDF metadata: First, as argued above, integrated

access to standard Web data in XML and metadata in RDF is essential. A framework to access RDF data through XML views is proposed. Second, this article argues that the currently predominant treatment of RDF data as flat triples is, although easy to comprehend, not the only and often not the best way of considering RDF data. Rather, a view of the RDF data directly as a graph is not only natural and closer to the RDF data model, but also allows for easy expression of graph patterns using much the same constructs as for navigating in XML data. This is particularly evident in face of incomplete information about the precise graph structure. Third, this article argues that querying RDF data is often most conveniently achieved if queries are composed in terms of both the triple and the graph view of RDF. Finally, this article argues that many applications call for queries combining object data in (X)HTML or XML and metadata in RDF. Thus, it is convenient to "marry" triple and graphs as well as RDF and XML in querying the Semantic Web.

The proposed framework is realized by rules in the XML query language Xcerpt that allow (a) the easy conversion between the two views on RDF and (b) the 'serialization transparent' querying of RDF, i.e., the querying of RDF in many of the over a dozen serialization formats for RDF proposed in recent years.

2 Preliminaries

2.1 RDF and RDF Schema: Metadata Representation in the Semantic Web

RDF [2] is the prevalent standard for representing metadata in the (Semantic) Web. RDF data is sets of 'triples' or 'statements' of the form (*Subject*, *Property*, *Object*). RDF's data model (as defined in [3]) is a directed graph, whose nodes correspond to statements' subjects and objects and whose arcs correspond to statements' property (thus relating subjects with objects). Nodes are labeled by either (1) URIs describing (Web) resources, or (2) literals (i.e. scalar data such as strings or numbers), or (3) are unlabeled, being so-called anonymous or 'blank nodes'. Blank nodes are commonly used to group or 'aggregate' properties. Edges are always labeled by URIs indicating the type of relation between its subject and object.

RDFS allows one to define so-called 'RDF Schemas' or 'ontologies', similar to object-oriented data models. Based on an RDFS, 'inference rules' can be specified, for instance the transitivity of the class hierarchy, or the type of an untyped resource that has a property associated with a known domain.

RDF can be *serialized* in various formats, the most frequent being XML. Early approaches to RDF serialization have raised considerable criticism due to their complexity. As a consequence, a surprisingly large number of RDF serialization have been proposed, cf. [4] for a survey of serialization formats.

Figure 1 shows the running example for this article, a (simplified) representation of an RDF graph as used, e.g., in a book recommender system.

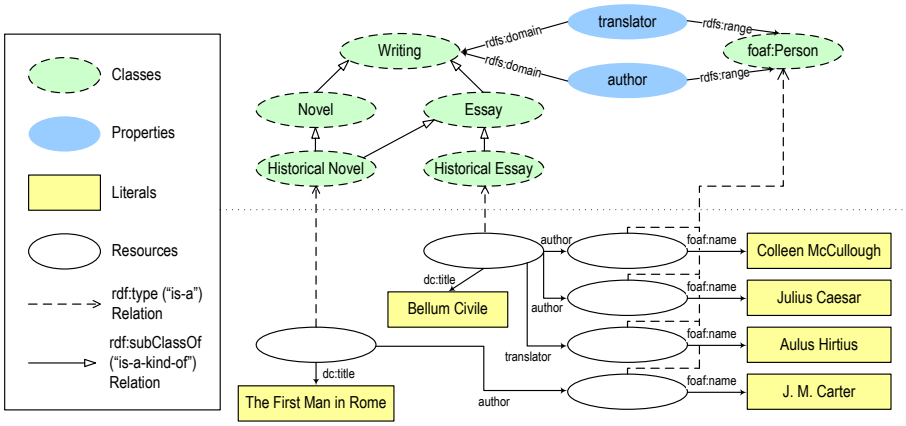

Fig. 1. Sample Data: representation as a (simplified) RDF graph.

2.2 Xcerpt, a versatile Web Query Language

Xcerpt [5, 6] is a query language designed after principles given in [7] for querying both data on the standard Web (e.g., XML and HTML data) and data on the Semantic Web (e.g., RDF, Topic Maps, etc. data).

Xcerpt is 'data versatile', i.e. a same Xcerpt query can access and generate, as answers, data in different Web formats. Xcerpt is 'strongly answer-closed', i.e. it not only gives rise to construct answers in the same data formats as the data queries, but also to include in a query program data generated by this same query program. Xcerpt's queries are pattern-based and give rise to incompletely specify the data to retrieve by (1) not explicitly specifying all children of an element, (2) specifying descendant elements at indefinite depths (restrictions in the form of regular path expressions being possible), and (3) specifying optional query parts. Xcerpt's evaluation of incomplete queries is based on a novel form algorithm called 'simulation unification'. Xcerpt's processing of XML documents is graph-oriented, i.e., aware of the reference mechanisms (e.g., ID/IDREF attributes and links) of XML. Xcerpt is rule-based: An Xcerpt rule expresses how data queried can be re-assembled into new data items.

Xcerpt programs consist of at least one 'goal' and some (possibly zero) 'rules'. Rules and goals contain query and construction patterns, called 'terms'. Terms represent tree- or graph-like structures. The children of a node may either be 'ordered' (as in a XHTML document or in RDF sequence containers), i.e. the order of occurrence is relevant, or 'unordered', i.e. the order of occurrence is irrelevant and may be ignored (as in the case of RDF statements). In the term syntax, an ordered term specification is denoted by square brackets `[ ]`, an unordered term specification by curly braces `{ }`. Terms may contain the *reference*

constructs `^id` ('referring' occurrence of the identifier `id`) and `id @ t` ('defining' occurrence of the identifier `id`). Using reference constructs, terms can form cyclic (rooted) graph structures.

Terms can be either data, query, or construct terms. *Data terms* represent XML documents and the data items of a semistructured database. They are similar to *ground* functional programming expressions and logical atoms. A *database* is a (multi-)set of data terms (e.g. the Web). A non-XML syntax has been chosen for Xcerpt to improve readability, but there is a one-to-one correspondence between an XML document and a data term.

Query terms are (possibly incomplete) patterns matched against Web resources represented by data terms. In many ways, they are like forms or examples for the queried data (in the style of the 'query-by-example' paradigm [8]), but also

- may be *incomplete in breadth*, i.e., contain 'partial' as well as 'total' term specifications: A term t using a partial term specification for its subterms matches with all such terms that (1) contain matching subterms for all subterms of t and that (2) might contain further subterms without corresponding subterms in t. Partial term specification is denoted by double (square or curly) brackets. In contrast, a term t using a total term specification does not match with terms that contain additional subterms without corresponding subterms in t.
- may be augmented by *variables* for selecting data items, possibly with 'variable restrictions' using the $\rightarrow$ construct (read **`as`**), which restricts the admissible bindings to those subterms that are matched by the restriction pattern.
- may contain *query constructs* like position matching (using **`position`**), subterm negation (using **`without`**), optional subterms (using **`optional`**), regular expressions for namespaces, labels, and text, and conditional or unconditional path traversal (using **`desc`**).
- may contain further constraints on the variables in a so-called *condition box*, beginning with the keyword **`where`**.

Construct terms serve to reassemble variables (the bindings of which are specified in query terms) so as to construct new data terms. Again, they are similar to the latter, but augmented by variables (acting as place holders for data selected in a query) and the grouping construct **`all`** (which serves to collect all instances that result from different variable bindings). Occurrences of **`all`** may be accompanied by an optional sorting specification.

Rules or construct-query rules relate a construct term to a query consisting of arbitrary boolean expressions using only **`AND`**, **`OR`**, and **`NOT`** to connect query terms. They have the form

```
CONSTRUCT query term FROM and { query term, or { query term, ... }, ... } END
```

An Xcerpt rule may contain one or several references to *resources* (expressed using **in** and **resource**).

Rules can be seen as 'views' specifying how to obtain documents shaped in the form of the construct term by evaluating the query against Web resources (e.g. an XML document or a database).

Xcerpt rules may be *chained* like active or deductive database rules to form complex query programs, i.e., rules may query the results of other rules.

3 Two Perspectives on RDF

This section introduces two different perspectives on RDF: (1) a flat, almost relational view and (2) a graph view reminiscent of semi-structured data. Existing approaches for RDF querying are classified along these perspectives briefly.

To illustrate these two perspectives, the selection query "Select all *Essays* together with their *authors* (i.e. author URIs and corresponding names)" is used against the data of Figure 1. This simple, but natural query requires a (unconditional) traversal of the sub-classes of Essay, to find also books classified as, e.g., *Historical Essay*.

3.1 RDF Triples: A Flat, Relational View

The following Xcerpt program expresses the above query on a triple view of the RDF data:

```
1  DECLARE ns-prefix rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   DECLARE ns-prefix books = "http://example.org/books#"
3  GOAL
     result [
5      all essay [
         id [ var Essay ],
7        all author [
           id [ var Author ],
9          all name [ var AuthorName ]
     ]  ]  ]
11 FROM
     and{ RDFS-TRIPLE [
13       var Essay, rdf:type{{}}, books:Essay{{}}            ],
       RDF-TRIPLE [
15       var Essay, books:author{{}}, var Author             ],
       RDF-TRIPLE [
17       var Author, books:authorName{{}}, var AuthorName    ]  }
   END
```

The query pattern (between **FROM** and **END**) is a conjunction of queries against the RDF triples represented in the predicate RDF-TRIPLE using the prefixes declared in line 1 and 2. Notice that the first conjunct actually uses RDFS-TRIPLE. This view of the RDF data contains all basic triples plus the ones entailed by the RDFS semantics (cf. [9] for a detailed description). Using RDFS-TRIPLE instead of RDF-TRIPLE ensures that also resources actually classified in a sub-class of books:Essay are returned.

In the construct pattern (between **GOAL** and **FROM**), one of the strengths of combining XML and RDF querying in Xcerpt is shown: Following the W3C's

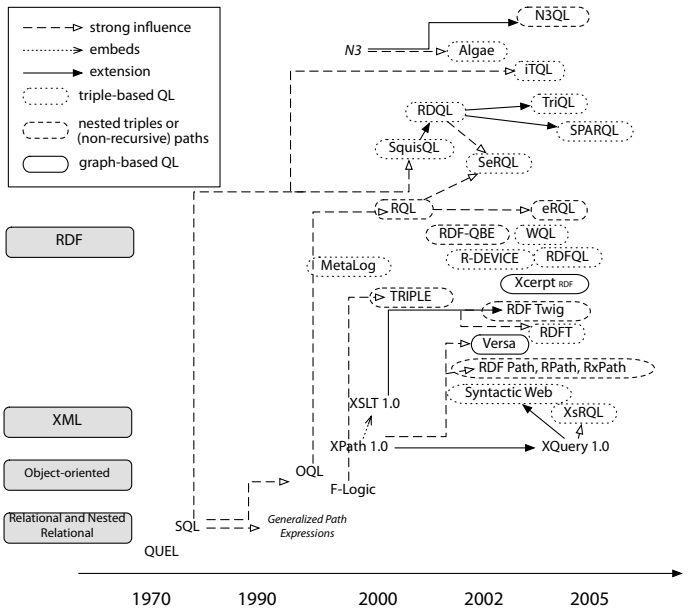

Fig. 2. RDF Query Languages: historical overview and classification.

requirements for an RDF data access language, yet in contrast to most other RDF query languages, it is possible to construct arbitrary XML: E.g., here, a list of all essays with their authors grouped inside is constructed. Indeed, when constructing structured data such as RDF and XML, grouping is among the most essential constructs, cf. [10, 11] on grouping in an XML context. For RDF querying, this points towards the need for similarly powerful, declarative, and explicit grouping constructs, as provide in Xcerpt's **`all`**.

Except for the construction of arbitrary XML, a similar (triple) view of RDF is taken in most of the current RDF query languages (cf. Figure 2), most notably in RDQL and the W3C's SPARQL [12], and also in [13], an approach for querying RDF with XQuery: A query is composed of conjunctions (and in some languages including our proposal disjunctions) of "triple patterns", i.e., triples with variables indicating queried data. Using multiple occurrences of same variables more complex conditions can be expressed, e.g., for traversing paths in the RDF data or even for restricting a resource using several of its properties.

While familiar from SQL, this style leads for RDF data to hard-to-read and lengthy queries that also pose problems for evaluation (cf., e.g., [14]). Furthermore, queries involving (conditional or unconditional) traversals of unknown length in the RDF graph can often not be expressed in query languages using this style and, if it is provided, requires recursive views, rules, or functions (e.g., in [13]). This applies to the traversal of the subclasses of *Essay* needed in the sample query. This is a serious limitation of triple-based RDF query languages, as

such queries are frequent (especially when considering ontological data in RDFS or other ontology languages) and recursive views or similar mechanisms make optimization and efficient evaluation of such queries hard or even impossible.

The previous observations lead us to an alternative view of RDF that is both closer to its actual data model and can make better use of the advanced features of an XML query language such as the traversal of arbitrary length paths in tree or graph data.

3.2 RDF Graphs: A Semi-structured View

For this view of RDF, Xcerpt's treatment of XML as graph data is an advantage over XML query languages such as XPath or XQuery, which consider XML as strictly tree shaped, providing no direct support for (ID/IDREF or similar) references in the data model. Although there have been proposals for slicing an (acyclic) RDF graph into trees (cf. Figure 2) for processing them with XSLT or XQuery (e.g., [15]), these approaches invariantly suffer (a) from choosing an appropriate slicing and (b) from the (in general) exponential blow-up of the tree view of an acyclic RDF graph.

In Xcerpt, a graph view of RDF is rather natural as the following Xcerpt program expressing the same query as above, but on the graph instead of the triple view, demonstrates:

```
DECLARE ns-prefix rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
DECLARE ns-prefix books = "http://example.org/books#"
GOAL
  result [
    all essay [
      id [ var Essay ],
      all author [
        id [ var Author ],
        all name [ var AuthorName ]
  ]  ]  ]
FROM
  RDFS-GRAPH {{
    var Essay {{
      rdf:type {{ books:Essay {{ }} }},
      books:author {{
        var Author {{ books:name {{ var AuthorName }} }}
      }}
    }} }}
END
```

The RDF graph view is represented in the RDF-GRAPH predicate. Here, the RDFS-GRAPH view is used that extends RDF-GRAPH as RDFS-TRIPLE extends RDF-TRIPLE. Triples are represented similar to striped RDF/XML: each resource is a direct child element in RDF-GRAPH with a sub-element for each statement with that resource as object. The sub-element is labeled with the URI of the predicate and contains the object of the statement. As Xcerpt's data model is a rooted *graph* this can be represented without duplication of resources.

In contrast to the previous query against the RDF triple view, no conjunction is used but rather a nested pattern that naturally reflects the structure of the RDF graph. The more complex a query, the more evident the advantage of the

graph view becomes: instead of having to use multiple occurrences of same variables for relating parts of the query, that relation is represented in the structure of the query itself (represented in the textual version of the query shown above by nesting and indentation).

Path traversals of arbitrary length can be expressed using traversal operators such as descendant. E.g., to find all subclasses of a given class one can use Xcerpt's qualified descendant `desc(rdfs:subClassOf<rdfs:Class)*` that is similar to regular path expressions or conditional XPath. Similarly, other constructs for querying XML data with incomplete information about the structure of the queried data can be used for RDF as well.

The following Xcerpt rule illustrate the use of a conditional descendant. It computes all persons that have a common ancestor and includes any such common ancestor, if it is the 'nearest' common ancestor, i.e., there is no other common ancestor on the path to the two persons. Since all persons have at least foaf:Person as common ancestor, the query also excludes all resources reached by rdf:type relations.

```
DECLARE ns-prefix rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
DECLARE ns-prefix books = "http://example.org/books#"
GOAL
  result [
    all related-persons [
      var Person1,
      var Person2,
      all via { var Resource }
    ]
  }
FROM
  RDFS-GRAPH {{
    var Resource @ /.*/ {{
      desc(!/rdf:type/)* var Person1 {{ rdf:type { foaf:Person {{}} } }},
      desc(!/rdf:type/)* var Person2 {{ rdf:type { foaf:Person {{}} } }},
      without desc /.*/ {{
        desc(!/rdf:type/)* var Person1 {{ }},
        desc(!/rdf:type/)* var Person2 {{ }},
      }}
    }}
  }}
END
```

Such explicit query constructs make optimization and evaluation of a frequent class of queries easier than relying on recursive views or similar generic mechanisms as required on the triple view. Considering the efficient evaluation of queries against such a graph view of RDF data, there are results on the efficient evaluation of queries against graph-shaped semi-structured data, cf. [16]. Ongoing work by the authors targets efficient evaluation methods for implementing Xcerpt queries against graph-shaped data. We believe it likely that at least for some interesting subsets of Xcerpt queries efficient evaluation methods against graph-shaped data can be found.

Nevertheless, only a surprisingly small number of RDF query languages consider a graph-view of RDF and provide expressive traversal operators, aside of Xcerpt most notably Versa [17, 18]. In contrast to the proposal presented in this paper, Versa uses an unfamiliar syntax instead of established traversal operators from XML and generalized path expressions.

Graph Merging. In contrast to conventional data such as XML or relational data, RDF data from different and heterogeneous sources can be easily merged, as nodes in an RDF graph can be and mostly (with the exception of blank and literal nodes) are identified by URIs, i.e., globally valid identifiers. On the first glance, it might seem that merging two RDF graphs is more difficult if considering the graph view of RDF. However, this crucial use case can be solved in Xcerpt easily on either view.

On parsing RDF data, Xcerpt annotates the RDF data with provenance information similar to recent proposals on named graphs [19] and their use in RDF query languages [20]: In the case of the triple view, a origin attribute is added to each RDF-TRIPLE term indicating the URI of the data's origin resource. In the case of the graph view, the same procedure could be taken. Alternatively, a single origin attribute for an entire RDF graph can be used by adding it to the RDF-GRAPH term. While this sacrifices some flexibility, it saves considerable space.

The following Xcerpt program shows the construction of the merged triples from the base triples. Notice, how the outer **all** groups only over the variables Subject, Property, and Object (as they occur free in that **all**, i.e., nested inside that **all** without another **all** in between). Therefore, a RDF-TRIPLE is created for each combination of subject, property, and object occurring in the base triples (from either graph) with a origin attribute that is a concatenation of the values of all origin attributes of base triples. If a statement occurred in both graphs the origin attribute will thus point to two resources.

```
CONSTRUCT
  merged-triples {{
    all RDF-TRIPLE [
      attributes {{ origin { all var Origin }, all var OtherAttributes }},
      var Subject, var Property, var Object
    ]
  }}
FROM
  RDF-TRIPLE [
    attributes {{ origin {{ var Origin }}, var OtherAttributes }}
    var Subject, var Property, var Object
  ]
END
```

If only named graph provenance (i.e., provenance for entire RDF graphs) is needed, the following rule illustrate the merging of two RDF graphs using the graph view:

```
CONSTRUCT
  RDF-GRAPH {
    all var Resources {{
      all var Statements {{ }}
    }}
  }
FROM
  and {
    RDF-GRAPH {{
      var Resource {{
        optional var Statements → /.*/ {{ }}
      }}
    }}
  }
```

```
15 END
```

Notice, how the query can simply ignore where the resources come from. Duplicate resources and statements are eliminated implicitly during the grouping.

3.3 Marrying Triples and Graphs

A final observation on the two views on RDF is that they are not mutually exclusive. In fact, conversion between the two views can be performed by the following, linear Xcerpt view:

```
1  CONSTRUCT
     RDF-GRAPH {
3      all var Subject @ var Subject {
         all optional var Predicate {  ^var Object   },
5        all optional var Predicate {  var Literal   }
       } }
7  FROM
     or{
9      RDF-TRIPLE[
         var Subject,  var Predicate{},
11       optional var Literal as literal{{}},
         optional var Object {{}} where { var Object != 'literal'}
13     ],
       RDF-TRIPLE[
15       /.*/:/.*/{{}},  /.*/:/.*/{{}},  var Subject{{}}
       ]  }
17 END
```

Notice the use of the **optional** keyword in lines 11 and 12. This indicates that the contained part of the pattern does not have to occur in the data, but if it does occur the contained variables are bound appropriately. In lines 3 and 4 the actual graph structure is constructed: by using the operators @ and ^ a (possibly cyclic) link can be constructed.

Indeed, the framework for RDF access in Xcerpt discussed in this article provides both views, thus allowing the query author to decide which view is more appropriate for his liking and requirements.

4 Marrying XML and RDF

Providing integrated access to RDF and XML is a crucial issue for the success of the Semantic Web. This is reflected by a number of proposals for such integrated access: As discussed above, [13] and [15] share with the work presented in this article the aim to extend XML query languages with access to RDF data, but are limited to a triple or tree view of RDF. In [21] the dual approach has been taken: mapping XML data into RDF. However, [21] only preserves a subset of the information represented in the XML data and requires schema-specific mapping rules to be defined prior to accessing the information. Reconciling the RDF and XML data models has been considered in [22] and [23]. Whereas the first essentially defines a new data model, the latter proposes a new node type for XML as means for handling RDF edges.

In the remainder of this section, the mapping from RDF into XML discussed in this article is further detailed.

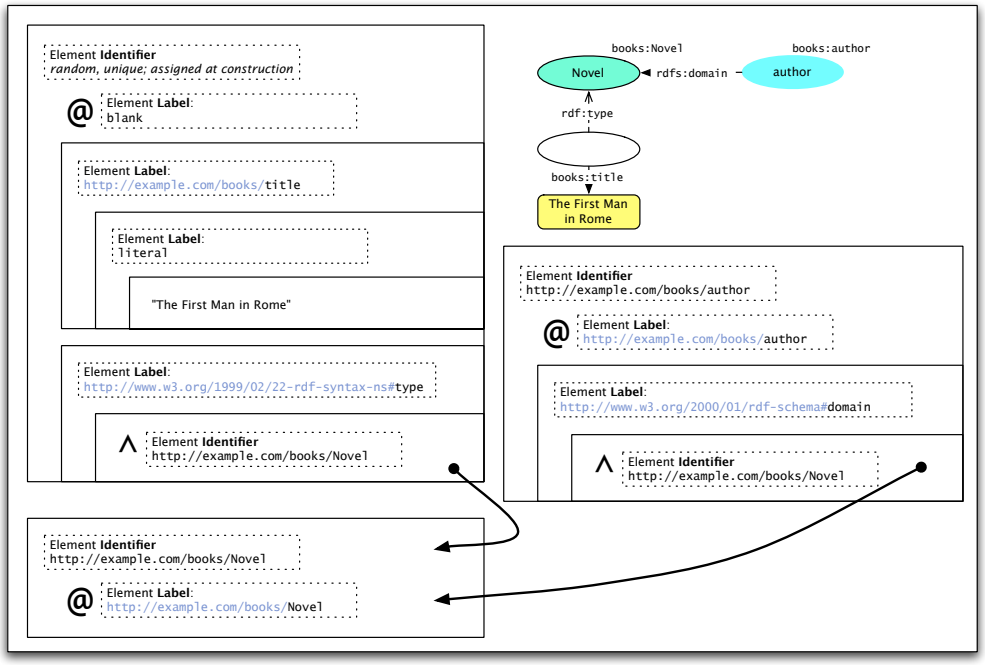

Fig. 3. Excerpt of RDF Graph represented in XML

4.1 A Marriage Contract: Issues when Mapping RDF into XML

The mapping proposed here, although it has some similarities with [23], differs from all of the above noticeably: Figure 3 illustrates the mapping from an excerpt of the sample RDF graph into three (top-level) XML elements (i.e., direct children of RDF-GRAPH). The following Xcerpt data term is a textual representation of the data in Figure 3:

```
DECLARE ns-prefix rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
DECLARE ns-prefix rdfs = "http://www.w3.org/2000/01/rdf-schema#"
DECLARE ns-prefix books = "http://example.org/books#"
RDF-GRAPH {
  id1 @ blank {
    books:title { literal { "The First Man in Rome" } },
    rdf:type { ^books:Novel }
  },
  books:author @ books:author {
    rdfs:domain { ^books:Novel }
  },
  books:Novel @ books:Novel {},
  ...
}
```

Notice how both edges and nodes from the RDF graph are represented as XML elements. However, nodes can still be distinguished as they are either blank nodes (without namespace) literal nodes (again without namespace) or named

resources in which case both their element identifier and element label are set to the URI identifying the resource. In contrast, elements for edges never have an identifier (as they can not be referenced by another part of the data). This mapping results in a 'stripped' representation of the RDF graph: the children of elements representing nodes (i.e., resources) are always elements representing edges and vice versa. One might question the use of resource URIs both as labels and identifiers of nodes. However, while element labels are more convenient for querying, unique identifiers for the elements are needed (for establishing graph references). Since URIs already provide uniqueness, they are also used for this purpose. [23] suggest the use of RDF types (i.e., the URI of the resource associated with rdf:type) as element labels when mapping RDF to XML. However, this approach is not able to map all RDF graphs as RDF resources may be classified by distinct types (that may not be related at all in the type hierarchy).

The XML mapping allows additional information about the RDF statements, e.g., provenance information, to be recorded alongside.

4.2 Serialization Transparency

Aside of providing the above discussed two views on RDF, Xcerpt's rules are also convenient for making the language 'serialization transparent'. For each RDF serialization, a set of rules expresses a translation from or into that serialization. Exemplary rules for RDF/XML and RXR can be found in [9], similar functions for parsing RDF/XML in XQuery are described in [13].

5 Conclusion and Outlook

In this article, a brief overview of a framework for RDF querying in the XML query language Xcerpt is presented highlighting in particular the need for reconsideration of the triple view as the only perspective on RDF available in the established RDF query languages. We believe that a richer view of RDF more akin to XML data with graph-shape not only makes the integration of data and metadata easier but also leads in many cases to more succinct queries without sacrificing efficiency.

Acknowledgments. This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web—A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American (2001)
2. Manola, F., Miller, E., McBride, B.: RDF primer. Recommendation, W3C (2004)

3. Klyne, G., Carroll, J., McBride, B.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C. (2004)
4. Bry, F., Furche, T., Badea, L., Koch, C., Schaffert, S., Berger, S.: Identification of Design Principles for a (Semantic) Web Query Language. Deliverable I4-D1, REWERSE (2004)
5. Schaffert, S., Bry, F.: Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In: Proc. Extreme Markup Languages. (2004)
6. Schaffert, S.: Xcerpt: A Rule-Based Query and Transformation Language for the Web. Dissertation/Ph.D. thesis, University of Munich (2004)
7. Bry, F., Furche, T., Badea, L., Koch, C., Schaffert, S., Berger, S.: Querying the Web Reconsidered: Design Principles for Versatile Web Query Languages. Journal of Semantic Web and Information Systems **1** (2005)
8. Zloof, M.M.: Query-by-Example: A Data Base Language. IBM Systems Journal **16** (1977)
9. Bolzer, O.: Towards Data-Integration on the Semantic Web: Querying RDF with Xcerpt. Diplomarbeit/Master thesis, University of Munich (2005)
10. Paparizos, S., Al-Khalifa, S., Jagadish, H.V., Lakshmanan, L.V., Nierman, A., Srivastava, D., Wu, Y.: Grouping in XML. In: EDBT Workshop on XML Data Management. Number 2490 in LNCS, Springer-Verlag (2002)
11. Beyer, K.S., Cochrane, R., Colby, L.S., Ozcan, F., Pirahesh, H.: XQuery for Analytics: Challenges and Requirements. In: Int. Workshop on XQuery Implementation, Experience and Perspectives <XIME-P/>. (2004)
12. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. Working Draft, W3C (2005)
13. Robie, J.: The Syntactic Web: Syntax and Semantics on the Web. In: XML. (2001)
14. Hung, E., Deng, Y., Subrahmanian, V.S.: RDF Aggregate Queries and Views. In: Int. Conf. on Data Engineering. (2005)
15. Walsh, N.: RDF Twig: accessing RDF graphs in XSLT. In: Extreme Markup Languages. (2003)
16. Schenkel, R., Theobald, A., Weikum, G.: HOPI: A Efficient Connection Index for Complex XML Document Collections. In: Extending Database Technology. (2004)
17. Olson, M., Ogbuji, U.: Versa Specification. Online only (2003)
18. Ogbuji, U.: Versa by example. Online only (2004)
19. Carroll, J., Bizer, C., Hayes, P., Stickler, P.: Named Graphs, Provenance and Trust. Technical Report HPL-2004-57, HP Labs (2004)
20. Bizer, C.: TriQL—A Query Language for Named Graphs. Online only (2004)
21. Koffina, I., Serfiotis, G., Christophides, V., Tannen, V., Deutsch, A.: Integrating XML data sources using RDF/S schemas: The ICS-FORTH Semantic Web Integration Middleware (SWIM). In: Dagstuhl Seminar on Semantic Interoperability and Integration. Number 04391 in Dagstuhl Seminar Proceedings, IBFI (2005)
22. Patel-Schneider, P., Simeon, J.: The Yin/Yang Web: XML Syntax and RDF Semantics. In: Int. World Wide Web Conference. (2002)
23. Boley, H.: The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. In: Int. Conf. on Applications of Prolog. (2001)