Let's Mix It: Versatile Access to Web Data in Xcerpt[*]

François Bry
Institute for Informatics,
University of Munich
Oettingenstrasse 67
81543 Munich, Germany
Francois.Bry@ifi.lmu.de

Tim Furche
Institute for Informatics,
University of Munich
Oettingenstrasse 67
81543 Munich, Germany
Tim.Furche@ifi.lmu.de

Benedikt Linse
Institute for Informatics,
University of Munich
Oettingenstrasse 67
81543 Munich, Germany
Benedikt.Linse@ifi.lmu.de

ABSTRACT

Applications and services that access Web data are becoming increasingly more useful and wide-spread. Web query languages provide efficient and effective means to access and process data published on the Web. Conventional Web query languages such as XQuery, XSLT, or SPARQL, however, focus only on one of the different data formats used on the Web and provide little to ease the integration of heterogeneous schemata or representations. Xcerpt is a *versatile* semi-structured query language, i.e., a query language able to access all kinds of Web data such as XML and RDF in the same language reusing common concepts and language constructs. In this article, we demonstrate how Xcerpt's features ease the integration of data sources that are heterogeneous in (a) data *format* (e.g., XML vs. RDF), (b) *schema*, and (c) concrete *representation* (if the schema allows representational variants). The results show that versatile Web query languages form a convenient foundation for data integration scenarios on the Web and suggest further consideration of their use in developing data-centric Web applications.

1. INTRODUCTION

Recent years have witnessed an upsurge in applications and services that access Web data: For instance, bibliography management applications access book data from Amazon, Barnes & Noble, and other vendors, citation data from CiteSeer, PubMed, ACM's digital library, etc., topic and researcher classifications in RDF format by crawling or from syndication sites, and keywords, abstracts, or table of contents from DocBook representations of articles.

Web query languages are tailored to the efficient and effective manipulation of such data. However, conventional Web query languages such as XQuery, XSLT, or SPARQL focus only on one of the different data formats available on the Web and provide little to ease the integration of the heterogeneous schemata or representations. We argue that *versatility*, i.e., the ability to handle in the same query program heterogeneous formats, schemata, and representations, is a crucial property of Web query languages, in particular to provide a convenient platform for the development of applications needing integrated access to different Web data sources. As defined in [6], a language is versatile if it provides means to handle heterogeneous data. One can distinguish three forms of versatility

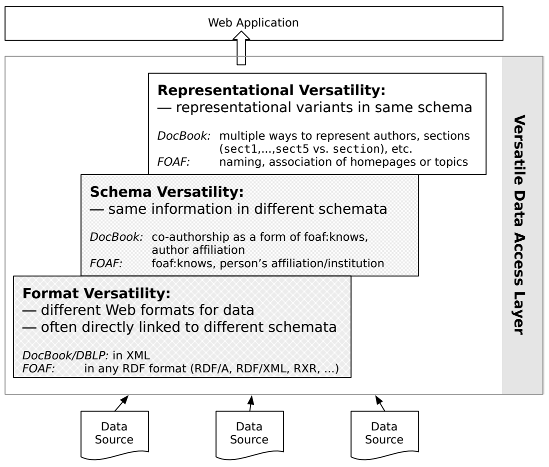
Web Application
Representational Versatility:
— representational variants in same schema
DocBook: multiple ways to represent authors, sections (sect1,...,sect5 vs. section), etc.
FOAF: naming, association of homepages or topics
Schema Versatility:
— same information in different schemata
DocBook: co-authorship as a form of foaf:knows, author affiliation
FOAF: foaf:knows, person's affiliation/institution
Format Versatility:
— different Web formats for data
— often directly linked to different schemata
DocBook/DBLP: in XML
FOAF: in any RDF format (RDF/A, RDF/XML, RXR, ...)
Versatile Data Access Layer
Data Source
Data Source
Data Source

Figure 1: Versatile Data Access on the Web

in Web query languages based on the origin of the heterogeneity that is considered by the language (summarized in Figure 1):

Format versatility: We call a language *format* versatile if it is able to query data in different (Web) formats such as XML, RDF, and Topic Maps. Whereas in the case of XML there is a precise mapping between serialization and data model, RDF and Topic Maps exhibit many different serialization formats, e.g., for RDF RDF/XML, RDF/A, and RXR (for a survey see [9]). The open and rapidly changing nature of the Web prevents that a small set of "built-in" formats suffices to achieve format versatility. If versatility beyond mere serialization differences is desired, the different data models have to be taken into consideration, cf. [11].

Schema versatility: Schema versatility is a slightly more elusive property of a query language linked to its ability to handle data represented according to different schemata. Ideally, the schema differences are handled transparently, e.g., by integrating the different schemata according to some given schema mapping. Such schema mappings may be specified, e.g., in RDFS, the RDF vocabulary description language, or in OWL, the Web ontology language. Direct or programmed support for schema mappings enables a query formulated according to one of the schemata to be evaluated against data in any of them.

Representational versatility: Web data is partially or semi-structured rather than fully structured like relational data. Therefore, even within the same schema, data may be represented in different ways, both w.r.t. structure and datatype. Web query languages should be able to take these representational variants into account, but in contrast to format and schema heterogeneity representational variants can not be handled transparently: For instance,

[*]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://www.rewerse.net/`).

WWW2006, May 22–26, 2006, Edinburgh, UK.
.

different ways to represent a section in DocBook (sect1 vs. section) carry additional semantics and must be distinguished in some contexts, whereas in many cases they can be considered the same. Such distinctions should be expressible in a versatile query language, allowing the programmer to choose the appropriate solution.

Xcerpt [15, 14] is one of the few Web query languages that address all three forms of versatility. It is a *semi-structured query language*, but very much unique among such languages: (1) In its use of a *graph data model*, it stands more closely to semi-structured query languages like Lorel than to recent mainstream XML query languages. (2) In its aim to address all *specificities of XML*, it resembles more mainstream XML query languages such as XSLT or XQuery. (3) In using (slightly enriched) *patterns* (or templates or examples) of the sought-for data for querying, it resembles more the "query-by-example" paradigm [17] than mainstream XML query languages using navigational access. (4) In its strict separation of querying and construction in rules it allows an easy transformation and interfacing of different rules. (5) In its use of rules as procedural abstraction or view mechanism, it provides a foundation for reasoning and mediation.

Following a short introduction to Xcerpt and the data formats considered, the remainder of the article further details the three forms of versatility along concrete examples realized in Xcerpt. Special emphasize is placed on the identification of general principles needed or useful for a versatile Web query language.

2. PROLEGOMENA

2.1 Web Formats Basics

Web data is currently, as far as it is not image, video, or layout-centric, mostly represented as semi-structured data, marked up either as XML (most often in the form of XHTML) or in one of the serializations of RDF (most often in the form of RSS). The discussion of versatile data access in Sections 3–5 uses data in both formats. To this end, the salient features of the two data representation formats are shortly summarized here.

XML [5] is a generic markup language for semi-structured data that has found widespread adoption both for data exchange and data representation on the Web (and beyond). Its data model is essentially a tree of nodes corresponding to elements (such as h1 or title in HTML) of the XML document. The tree structure reflects the nesting of elements in the serial XML document. Elements may contain text content represented as text node children in the data model. Other features of XML include attributes, namespaces, and processing instructions, for details see [7].

Where XML data is used in the following sections, the examples are mostly drawn from a list of articles, papers, conferences, etc. in the style of DBLP[1], but with additional information about the actual content of the paper in DocBook format[2]. Figure 2 shows a visual representation of parts of the sample XML document (with fictional journal information) using Xcerpt's visual companion language visXcerpt [3].

RDF [10] 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 is a directed graph, whose nodes correspond to subjects and objects of statements and whose arcs correspond to their properties relating subjects and objects. Nodes are labeled by either (1) URIs describing (Web) resources, or (2) literals (i.e., scalar data such as strings

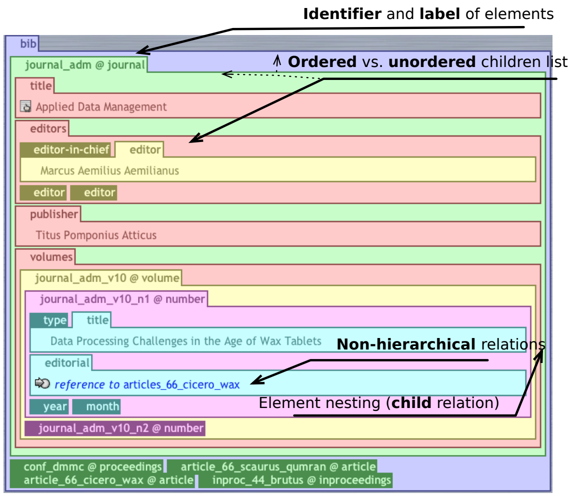
Identifier and label of elements
Ordered vs. unordered children list
bib
journal_adm @ journal
title
Applied Data Management
editors
editor-in-chief
editor
Marcus Aemilius Aemilianus
editor
editor
publisher
Titus Pomponius Atticus
volumes
journal_adm_v10 @ volume
journal_adm_v10_n1 @ number
type
title
Data Processing Challenges in the Age of Wax Tablets
Non-hierarchical relations
editorial
reference to articles_66_cicero_wax
year
month
Element nesting (child relation)
journal_adm_v10_n2 @ number
conf_dmmc @ proceedings
article_66_scaurus_qumran @ article
article_66_cicero_wax @ article
inproc_44_brutus @ inproceedings

Figure 2: Visual Rendering of Sample XML Data

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 Schemata" or ontologies, similar to object-oriented data models. Based on RDFS, inference rules can be specified, for instance the transitivity of the class hierarchy.

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. [9] for a survey of serialization formats.

In the following, example data based on the "Friend of a Friend" (FOAF) project[3] is used. FOAF is an RDF vocabulary describing mostly foaf:Persons by properties such as foaf:name, foaf:mbox, foaf:homepage, foaf:interest, etc. Furthermore, it allows to establish "social networks" of persons using foaf:knows, foaf:Project, foaf:Group, foaf:Organization, and foaf:member.

2.2 XcerptBasics

As introduced above, Xcerpt is a query language designed after principles given in [6] for querying both data on the standard Web and data on the Semantic Web. More information, including a prototype implementation, is available at `http://xcerpt.org`.

2.2.1 *Data as Terms*

Xcerpt uses **terms** to represent semi-structured data. *Data terms* represent XML documents, RDF graphs, and other semi-structured data items. Notice that subterms (corresponding to, e.g., child elements) may either be "ordered" (as in an 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 (possibly cyclic, but rooted) graph structures. Term attributes are denoted in round parentheses. Terms are similar to ground functional programming

[1]`http://www.informatik.uni-trier.de/~ley/db/`
[2]`http://www.oasis-open.org/docbook/`

[3]`http://www.foaf-project.org/`

expressions and logical atoms. 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.

2.2.2 Queries as Terms

Following the "Query-by-Example" [17] paradigm, queries are merely examples or *patterns* of the queried data and thus also terms, annotated with additional language constructs. Xcerpt separates querying and construction strictly.

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, 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. Query terms may further be augmented by *variables* for selecting data items, possibly with "variable restrictions" using the $\rightarrow$ construct, which restricts the admissible bindings to those subterms that are matched by the restriction pattern. They may contain *query constructs* like **position** matching, subterm negation using **without**, **optional** subterms, regular expressions for namespaces, labels, and text, and conditional or unconditional path traversal using **desc**. Finally, they 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 gained from the evaluation of 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.

2.2.3 Rules and Programs

Query and construct terms are related in **rules** which themselves are part of Xcerpt **programs**. Rules have the form:

```
CONSTRUCT construct-term
FROM and { query-term or { query-term ... } ... } END
```

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. More details on the Xcerpt language and its syntax can be found in [14, 15].

3. FORMAT VERSATILITY

The most basic type of versatility a query language for the (Semantic) Web should posses, is *format versatility*. Data on the Web is encountered in many different XML markup languages. In contrast, metadata is usually represented in RDF or Topic Maps. However, XML serializations exist for both of these meta data standards, which makes their integration with XML data easier. Therefore, it has been proposed, e.g., in [13], that an ordinary XML query language such as XQuery already provides all necessary means to integrate data from these different formats. However, using ordinary XML query languages for such an integration proves to be infeasible for a number of reasons:

(1) *Limitations of the XML Data Model:* The W3C's data model for XML, the XML Infoset [7], deviates from most other semi-structured data models (including those of RDF and Topic Maps) in two notable ways: it is tree-shaped, handling non-hierarchical relations as "second class" relations, and it assumes that the order among the children of a node is always relevant. These limitations are true neither for RDF nor Topic Maps and make format versatile extensions of ordinary XML query languages such as XQuery or XSLT difficult at best: Either the approach has to tackle "slicing" up an RDF or Topic Maps graph in XML trees [16], leading to very unnatural queries, or relational representations are used to represent RDF triples or Topic Maps assertions, e.g., in [13].

(2) *Multitude of Serialization Formats:* Topic Maps and, to an even greater degree, RDF exhibit numerous structured text and XML serialization formats, e.g., the W3C syntax for RDF RDF/XML [2], a syntax for embedding RDF in arbitrary XHTML documents RDF/A [1], and Turtle, the RDF triple syntax adopted for W3C's SPARQL. A format versatile language should thus be able to adopt to rapidly emerging serialization variants.

(3) *Transparent Integration:* Integrating data from different formats with ordinary XML query languages can be quite cumbersome and unnatural, as the integration must be performed as part of each query accessing the integrated data. Xcerpt rules represent a high level construct that can be used to provide a uniform logical view over XML markup languages and RDF serializations.

On the other hand, existing RDF and Topic Maps query languages are equally unsuited for XML processing, partially due to the limited expressiveness of most of these languages (including the W3C's SPARQL), partially due to the specificities of XML related to its document markup origins not considered in these languages.

Therefore, a proper versatile query language is called for that has transparent support for different formats and a data model capable of seamlessly, but without loss of (relevant) information, integrating semi-structured data on the Web represented in either XML, RDF, or Topic Maps.

The remainder of this section illustrates this point along a number of examples realized in Xcerpt. The above mentioned data sources, a DBLP-like article collection and FOAF data are used.

3.1 From XML to RDF

Xcerpt is tailored in many aspects closely to XML, making the access to XML data like speaking ones "mother tongue". In this section we show how the co-author relation (indicated by the name dblp:coauthor) can be extracted from the DBLP-like article collection. Consider the following fragment:

```
bib(){
 article.66.cicero.wax @ article(){
  authors()[
   author()[ name(){ "Marcus Tullius Cicero" }
        affiliation()[ "Governor, Cicilia" ] ]
   author()[ name(){ "Marcus Aemilius Lepidus" } ] ]
  title()[ "Data Storage on Wax Tablets" ] }
```

From this XML fragment it can be deduced that there are persons with the names "Marcus Tullius Cicero" and "Marcus Aemilius Lepidus" who are co-author of each other. XML does not associate unique identifiers with the elements in a document. Hence, blank nodes must be used to represent these persons in the corresponding RDF graph. The rules below transform the DBLP-like article collection to a set of RDF-like triples containing all dblp:coauthor predicates for authors of the same paper. For simplicity, it is assumed that names are unique within the article collection. If this assumption does not hold, additional properties of the author such as his affiliation could be leveraged to resolve such name conflicts in some cases.

Two queries are needed to extract this information from the article collection. First we need to establish a b-node for each *distinct* author name:

```
CONSTRUCT
 DISTINCT-NAMES{ all distinct name[var AName] }
FROM
 bib{{ _ {{ # Article, Inproceeding, Techreport, ...
        desc author{{ name{ var AName } }} }} }}
END
```

An author name is found in the XML article collection in name children of author elements under top-level entries representing articles, theses, technical reports, etc. Since there may be elements between entry and author (e.g., authorgroup or affiliation) Xcerpt's **desc** modifier is used to indicate that authors at any depth are to be included. In the construct part of the rule, a temporary store for the names is created that contains one child for each **distinct** binding of the variable AName.

In a second rule, we query the result of the first one: For each pair of authors within the same top-level entry (i.e., publication) the (automatically assigned) ID of the corresponding name element as created by the previous rule is queried and used as the local ID of the b-node for that author.

```
CONSTRUCT
 RDF-STORE{
  all ( triple[value(idvar AID1), "dblp:coauthor",
             value(idvar AID2)] ) }
FROM
 and (
  DISTINCT-NAMES{{
   idvar AID1 @ name[var AName1]
   idvar AID2 @ name[var AName2] }}
  bib{{ /.*/{{
        desc author{{ name{ var AName1 } }}
        desc author{{ name{ var AName2 } }} }} }} )
END
```

3.2 Normalizing RDF or From RDF to RDF

As mentioned above, a multitude of serializations for RDF in XML and structured text exists and frequently new serialization formats are adopted. Therefore, built-in support for one or even a small number of these serialization formats alone is not sufficient. Alternatively or additionally, a versatile Web query language should provide user definable mappings from arbitrary RDF serializations to a uniform or normalized view of RDF used as target for queries. In other words, the various *physical representations* of RDF must be mapped to a *logical view* of RDF.

This section illustrates such mappings as Xcerpt rules for some of the available RDF serialization, viz. RDF/XML [2], and RDF/A [1]. Fortunately, this can be achieved independently of the schema of the RDF data, which means that the examples of this section work just as well for any other vocabulary than FOAF.

In a semi-structured query language two possible *logical views* of RDF are most reasonable: RDF as (relational) triples and as proper graph.[4] For many queries the second view is more favorable, as it allows the leveraging of expressive graph traversal operators such as **descendant** or regular path expressions in queries. However, for simplicity a triple view of RDF as in the above examples is assumed in the following. For a more detailed discussion of choosing an appropriate logical view on RDF see [8].

3.2.1 Transforming RDF/XML to Triples

The standard serialization format for RDF is RDF/XML [2], a W3C recommendation since 2004 very close to the original 1999 RDF syntax. Surprisingly, it is very difficult to parse this serialization format as it has a high degree of variability. This originates partially from the design goal that the syntax allows terse statements of large XML graphs without unnecessary repetition or duplication in the syntax leading to a large number of abbreviations and purely syntactical variants, making reading and processing of RDF/XML non trivial. The following example document shows a few fictive statements in RDF/XML about "Marcus T. Cicero" based on the FOAF vocabulary:

```
<?xml version="1.0" encoding="utf−8"?>
<rdf:RDF ... >
 <jur:Lawyer rdf:about="people:m_t_cicero"
        foaf:name="Marcus T. Cicero">
  <foaf:member rdf:resource="pol:Optimates" />
  <foaf:depiction>
   <rdf:Description>
    <foaf:creator
       rdf:resource="people:m_t_cicero" />
   </rdf:Description>
  </foaf:depiction>
 </rdf:Description></rdf:RDF>
```

Description elements represent resources occurring as subjects in RDF triples. They contain elements or attributes that define their properties. The object of a statement is attached as attribute value, as element content, or as value of the special rdf:resource attribute. Thus, the above RDF/XML document defines the following RDF triples (in Turtle notation).

```
people:m_t_cicero foaf:name "Marcus T. Cicero".
people:m_t_cicero rdf:type <jur:Lawyer>.
people:m_t_cicero foaf:member pol:Optimates.
people:m_t_cicero foaf:depiction _:bust_17.
_:bust_17 rdf:creator people:m_t_cicero .
```

This example gives only a glimpse at the many variants allowed in RDF/XML. For more details on the variants and a full description on how to use Xcerpt to transform RDF/XML in a triple view of RDF are given in [4].

A brief look at one of the transformation rules suffices to demonstrate the level of versatility needed to integrate such formats:

```
CONSTRUCT
 RDF-STORE{ all triple[var SURI, var PURI, var OURI] }
FROM
 and(
  rdf-subjects {{
   idvar S @ _{{ var PURI (( rdf:resource=var OURI )){{
   }} }} }},
  node-to-triple-value[ idvar Subject, var SubjURI ] )
END
```

The rule uses two helper rules rdf-subjects and node-to-uri to find all subject resources in the RDF/XML document (this requires a recursive traversal of the document, as subject resources may occur at any depth and are only distinguishable from properties and objects through their structural position). The second helper rule is node-to-uri that associates nodes in the RDF/XML document with URIs (needed to resolve relative URIs, assign "URIs" to blank nodes etc.). The above rule selects for each subject node the immediate children of that node, which represent the properties of the subject node. Finally, the URI of the object is selected from the value of the rdf:resource attribute of the property. Obviously, this rule only covers one of the many cases how triples are represented, it can, e.g., not handle literal objects or nested objects.

3.2.2 Transforming RDF/A to Triples

RDF/A [1] is a recent W3C editor's draft proposing a new serialization of RDF that allows to embed RDF statements as attributes in any possible XML markup language, such as XHTML or SVG. An example RDF/A fragment is shown in the following listing.

```
<p about="http://senate.spqr/m_t_cicero">
```

[4]Obviously, any structure in between could also be chosen, however, as [16] shows it is far from obvious to choose a good "slicing" of the RDF graph that determines which relations are expressed through direct links and which through value references.

```
For many years, <span property="foaf:name">Marcus T.
Cicero</span> is a recognized name, both as a <span
property="rdf:type" href="jur:Lawyer">lawyer</span> and as
a senator. He is a member of the conservative <a
property="foaf:member" href="pol:Optimates">Optimates
party</a>. He has also created <span href="[_:bust_17]"
rev="foaf:depiction" rel="foaf:creator">Bust 17</span>
depicting himself.</p>
```

This RDF/A fragment represents the same triples as the above RDF/XML document: Subjects of statements are indicated by the about attribute, predicates by one of the attributes property (if the object of the statement is a literal), rel (if the object of the statement is a URI) and rev (if the statement is to be read in the reverse direction). In case the objects of statements are literals, in RDF/A they are either included in the element with the subject and predicate attributes or in a content attribute, which takes precedence. If the object is an URI, it is included in an href attribute.

The different ways RDF triples may be embedded in XHTML (or any other XML markup language) can be covered in the disjuncts of a single Xcerpt rule:

```
CONSTRUCT
RDF-STORE{ all triple[ var S, var P, var O] }
FROM
or (
 desc _((about=var S, property=var P, without content=_))
      {{var Object}}
 desc _((about=var S, rel=var P, href=var O)){{ }}
 desc _((about=var O, rev=var P, href=var S)){{ }},
 desc _((about=var S, property=var P, content=var O)){{}})
END
```

Not all possible embeddings of triples in the RDF/A syntax are covered by this rule: Is the about attribute absent for an element with a property attribute, the subject of the corresponding statement is resolved by *subject resolution*, cf. Section 5.

There are also other, non-W3C RDF serialization formats that are more regular and become very similar to the logical triple view of RDF discussed in this section.

3.2.3 From Triples to Graphs

Given the triple view, one can formulate easily expressive queries against the RDF data. However, whenever the queries involve path traversals, in particular arbitrary length path traversals (e.g., to traverse the transitive closure of a relation) complex and often recursive rules are needed.

However, if RDF is considered as a graph, where similar as in RDF/XML subjects contain properties which in itself contain links to objects of statements, then such queries can be expressed with **descendant** or regular path expressions as available in most XML and semi-structured query languages.

In the spirit of versatility, Xcerpt provides access to both logical views. The following rule transforms the triple into a graph view.

```
CONSTRUCT
 RDF-GRAPH-STORE {
  all var Subject @ var Subject {
   all optional var Predicate { ^var Object },
   all optional var Predicate { var Literal } } }
FROM
 RDF-TRIPLE-STORE[
  triple[ var Subject, var Predicate,
         optional var Literal →literal{{}},
         optional var Object →/.*/ ] ]
END
```

3.3 From Topic Maps to RDF

Topic Maps being an ISO standard fitting similar purposes as RDF, it is often desirable to draw information from both of these semantic Web data formats simultaneously. The large amount of research aiming at easing the interoperability amongst both formats, cf. [12], is an indicator for the necessity of a versatile query language like Xcerpt that allows the aggregation of information from both formats.

A possible procedure for integrating Topic Maps with other formats would be the transformation of topics and associations to sets of triples in a similar way as the transformation of the DBLP-like article collection above. There are several kinds of triples that may be extracted from a topic map: the type of the topic, its name, and named occurrences of the topic. The transformation of topics to RDF-like triples—and therefore the integration of information from Topic Maps and RDF—is obtained quite naturally using a format versatile query language and is therefore omitted for space reasons.

4. SCHEMA VERSATILITY

Schema versatility builds upon format versatility in the sense that the integration of information from different resources in many cases requires that the employed query language is both format versatile and schema versatile. In fact, it is unlikely that pieces of information gathered from sources in different formats make use of the same schema. As an example reconsider the integration of information from the DBLP-like article collection and FOAF descriptions. While both sources of information have been brought into a uniform triple notation thanks to format versatility, the schemata of both sources remain unassociated. This is where schema mappings specified, e.g., in the W3C's RDFS vocabulary definition language or in the Web Ontology Language (OWL) come into play.

Both languages provide some means to establish a mapping between classes and properties in different schemata, though the mapping concepts of RDFS are very limited. A schema mapping might, e.g., state that dblp:coAuthorOf is a subproperty of foaf:knows, i.e., that all resources that stand in dblp:coAuthorOf relation also stand in foaf:knows relation.

If the query language supports reasoning with RDFS and/or OWL, it suffices to include such schema mappings into the considered data. The query engine then infers the appropriate tuples. Xcerpt provides such reasoning support for RDFS in form of a rule library that also illustrates how user defined schema mappings (e.g., going beyond the mapping constructs of either RDFS or OWL) can be supported in a query language. Xcerpt's RDFS rule library is described in more detail in [4, 8]. Here, it suffices to give an impression of the kind of rules needed to realize RDFS reasoning:

```
CONSTRUCT
 RDFS-STORE{
  optional all triple[var Subject, var SuperPr, var
      Object]
  optional all var BasicTriple }
FROM
 or (
  RDF-STORE{{
   triple[var SubPr, "rdfs:subPropertyOf", var SuperPr]
   triple[var Subject, var SubPr, var Object] }}
  RDF-STORE{{ var BasicTriple }} )
END
```

The rule queries the RDF-STORE for all triples with predicate rdfs:subPropertyOf. Such triples connect a sub-property SubPr to a super-property SuperPr. In the inferred RDFS-STORE a new triple is inserted for each basic triple with the sub-property as predicate. Additionally, all the basic triples are included as well.

Beyond simple equivalences or specialization relations, schema mappings may contain more elaborate information, e.g., that a property of an object is a primary key, i.e., its values uniquely identify that object. In OWL this is specified by typing the property as owl:InverseFunctionalProperty. This information can be used to recognize that two objects, even if they are identified differently (most commonly at least one of them is a blank node) are indeed the same. In the example case, ISBNs or DOIs of books and articles qualify for inverse functional properties. This allows to infer equivalence of individual books even if the schema mapping contains only equivalences on properties and classes.

Schema versatility often goes hand in hand with the two other forms of versatility: Often different schemata originate from different formats used for the data; often different schemata use different representations for the same data. The link between schema and representational versatility is further investigated in the following section.

5. REPRESENTATIONAL VERSATILITY

Even within the same schema, the represenation of information may vary to a great extent, and the semi-structured nature of data on the Web requires that an adequate query language can handle heterogeneous and incomplete data and complex nested structures. RDF/A is an example for an XML schema that allows a great degree of representational diversity. Especially the concept of subject resolution, which has been mentioned in Section 3, requires that a Web query language that is supposed to handle RDF/A is representationally versatile. Subject resolution in RDF/A means that in the absence of an about attribute, the subject of a statement is searched for as the value of the nearest available about attribute of enclosing elements.

In order to extract also triples of this kind, the rule from Section 3 must be adjusted to use Xcerpt's regular path expressions. The second disjunct of the above rule would read as follows.

```
desc _ (( about=var Subject )){
 desc(!(_[about=_]))*
   (( rel=var Predicate, href=var Object )){{ }}
}}
```

The FOAF vocabulary specification provides many different ways to specify the name of a person, such as foaf:firstName, foaf:nick, foaf:givenname, foaf:family_name, foaf:name, and foaf:surname. The automatic creation of an address book from a set of FOAF descriptions requires that all or some of these possibilities are taken into account. Undoubtedly, a representationally versatile query language must provide a construct that allows certain parts of semi-structured data to be *optional*, in the sense that they are to be retrieved if present, but that the query need not fail if they are absent. The following Xcerpt rule transforms a set of FOAF descriptions into an address book making intensive use of the **optional** construct.

```
CONSTRUCT
 addressbook[
  all address[
    mbox[ var Mbox ],
    firstname[ optional var FirstName, optional var
          GivenName ],
    familyname[ optional var FamilyName, optional var
          Surname ],
    optional name[ var Name ] ] ]
FROM
 foaf:Person{{
  foaf:mbox{ var Mbox },
  optional foaf:firstName{ var FirstName },
  optional foaf:family_name{ var FamilyName },
  optional foaf:surname{ var Surname },
  optional foaf:name{ var Name },
  optional foaf:givenname{ var GivenName } }}
END
```

In construct terms, **optional** marks the enclosed subterms as optional, i.e., they are only included in the result, if there are bindings for the free variables in the scope of the **optional**. Therefore, the element name in the above rule is only included, if the query part of the rule succeeded to match the Name variable.

Note that the rule is written based on the assumption that there is no major semantic difference neither between surnames and family names, nor between first names and given names. Furthermore, it is assumed that a single FOAF description does not contain both a family name and a surname or a first name and a given name. If this is not the case, a precedence could be given to, e.g., firstName and familyName and realized with Xcerpt's conditional construction.

6. CONCLUSION

We believe that query languages are the right tools for many applications to access Web data and can also provide flexible, rich interfaces if used for publishing Web data. However, on the Web data access is not limited to one format, to one schema, or to one representation as can be assumed for many traditional usage scenarios of query languages. Therefore, query languages for the Web have to be able to deal with heterogeneity at all levels to become truly useful for application developers. Xcerpt is, in our opinion, a first step towards realizing this vision of versatile query languages that make access to heterogeneous data sources almost as easy as access to homogeneous data.

7. REFERENCES

[1] B. Adida and M. Birbeck. RDF/A Primer 1.0—Embedding RDF in XHTML. Internal draft, W3C, 2006.

[2] D. Beckett and B. McBride. RDF/XML Syntax Specification (Revised). Recommendation, W3C, 2004.

[3] S. Berger, F. Bry, O. Bolzer, T. Furche, S. Schaffert, and C. Wieser. Xcerpt and visXcerpt: Twin Query Languages for the Semantic Web. In *Proc. Intl. Semantic Web Conf.*, 2004.

[4] O. Bolzer. Towards Data-Integration on the Semantic Web: Querying RDF with Xcerpt. Master thesis, University of Munich, 2005.

[5] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (3rd Ed.). Recommendation, W3C, 2004.

[6] F. Bry, T. Furche, L. Badea, C. Koch, S. Schaffert, and S. Berger. Querying the Web Reconsidered: Design Principles for Versatile Web Query Languages. *J. of Semantic Web and Inf. Sys.*, 1(2), 2005.

[7] J. Cowan and R. Tobin. XML Information Set (2nd Ed.). Recommendation, W3C, 2004.

[8] T. Furche, F. Bry, and O. Bolzer. Marriages of Convenience: Triples and Graphs, RDF and XML. In *Proc. Intl. Workshop on Principles and Practice of Semantic Web Reasoning*, 2005.

[9] T. Furche, F. Bry, S. Schaffert, R. Orsini, I. Horrocks, M. Krauss, and O. Bolzer. Survey over Existing Query and Transformation Languages. Deliverable I4-D1, REWERSE, 2004.

[10] F. Manola, E. Miller, and B. McBride. RDF Primer. Recommendation, W3C, 2004.

[11] P. Patel-Schneider and J. Simeon. The Yin/Yang Web: XML Syntax and RDF Semantics. In *Proc. Intl. World Wide Web Conf.*, 2002.

[12] S. Pepper, F. Vitali, L. M. Garshol, N. Gessa, and V. Presutti. A Survey of RDF/Topic Maps Interoperability Proposals. W3C, 2006.

[13] J. Robie. The Syntactic Web. In *Proc. XML Conference and Exhibition*, 2001.

[14] S. Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web.* Ph.D. thesis, University of Munich, 2004.

[15] S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proc. Extreme Markup Languages*, 2004.

[16] N. Walsh. RDF Twig: accessing RDF graphs in XSLT. In *Proc. Extreme Markup Languages*, 2003.

[17] M. M. Zloof. Query By Example: A Data Base Language. *IBM Systems Journal*, 16(4):324–343, 1977.