Data Model and Query Constructs for Versatile Web Query Languages: State-of-the-Art and Challenges for Xcerpt

François Bry, Tim Furche, and Benedikt Linse

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

Abstract. As the Semantic Web is gaining momentum, the need for truly versatile query languages becomes increasingly apparent. A Web query language is called versatile if it can access in the same query program data in different formats (e.g. XML and RDF). Most query languages are not versatile: they have not been specifically designed to cope with both worlds, providing a uniform language and common constructs to query and transform data in various formats. Moreover, most of them do not provide a flexible data model that is powerful enough to naturally convey both Semantic Web data formats (especially RDF and Topic Maps) and XML. This article highlights challenges related to the data model and language constructs for querying both standard Web and Semantic Web data with an emphasis on facilitating sophisticated reasoning. It is shown that Xcerpt's data model and querying constructs are particularly well-suited for the Semantic Web, but that some adjustments of the Xcerpt syntax allow for even more effective and natural querying of RDF and Topic Maps.

1 Introduction

Data on the web is increasingly enriched with semantic meta-data, linking it to the real world or to other information. While XML has already gained widespread acceptance, RDF is on the best way to do so. Query languages have established themselves as a valuable means for accessing both formats, and a considerable number of query languages for XML (such as XQuery[1], XPath[2], XSLT[3], Xcerpt[4–6]) and for Semantic Web data (e.g. SPARQL[7], RQL[8], Versa[9]) have been proposed and implemented, cf. [10] for a survey. XML query languages can be used to query XML serializations of RDF data. This, however, hardly yields a programmer-comfortable approach to RDF data. In fact, most of the above languages have not been specifically designed to cope with both worlds, and do not provide a uniform language and common constructs to query and transform data in the various formats. Moreover, most of them lack a flexible data model that is powerful enough to naturally comprehend both Semantic Web data formats (especially RDF and Topic Maps) and XML.

This article highlights challenges related to the data model and convenient constructs for querying both standard Web and Semantic Web data with an emphasis on facilitating sophisticated reasoning. It is shown that Xcerpt's data model and querying constructs are well-suited also for the Semantic Web, but that some adjustments of Xcerpt's syntax would allow for even more effective and natural query authoring with respect to RDF and Topic Maps.

The rest of this article is structured according to its contributions: Section 2 examines requirements related to the data model of versatile web query languages with focus on RDF and XML. Section 3 proposes an extended edge-labeled syntax for Xcerpt terms that can be straightforwardly mapped to usual Xcerpt data terms. Section 4 illustrates that Xcerpt's constructs for handling heterogeneity are beneficial to both XML and RDF querying. Section 5 underlines the importance of grouping constructs in the scope of the Semantic Web. Finally, Section 6 concludes this article and sheds light upon further research both with respect to the language itself and its efficient evaluation.

2 Challenges Related to the Data Model

Figure 1 presents two possible representations of information about countries, their names and their border-countries in XML (on the left hand side) and RDF (on the right hand side). Nodes of the XML document tree are represented as grey rectangles containing the element name. Text nodes are are distinguished by quotes and attribute-value pairs are displayed at the top right of the node they belong to. The namespace prefixes `rdf`, `rdfs` and `geo` are assumed to be bound to `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, `http://www.w3.org/2000/01/rdf-schema#` and `http://geo.org/terms#`, respectively throughout this article. Nodes of the RDF graph on the right are either depicted as grey rectangles containing the URI or blank node name in the case of non-literals or as orange nodes in the case of literal values.

Figure 1 naturally exemplifies that XML semi-structured data and Semantic Web data differ in various ways, complicating the conversion of the formats in either direction and impeding the use of a query language specialized on only one of the formats for accessing both. On the one hand, XML data can hardly be transformed to RDF, because (1) the order of outgoing edges in RDF is irrelevant, (2) nodes are uniquely identified by URIs except for literals and blank nodes, (3) RDF does not support the concept of attributes. On the other hand, XML cannot naturally comprehend RDF data, in that (1) besides nodes also the edges of RDF graphs are labeled, (2) RDF is truly graph structured, and (3) RDF graphs need not be connected and are unrooted. In this section all of these differences are discussed and it is illustrated that although Xcerpt data terms are purely node-labeled, they can represent RDF data in a very straight-forward way.

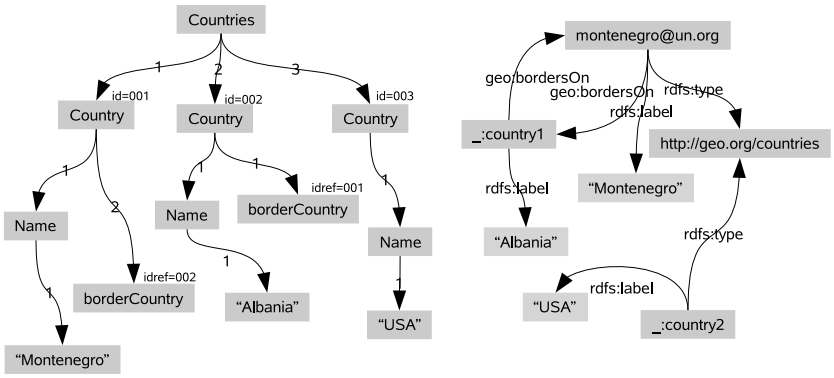

Fig. 1. XML data versus RDF data

2.1 Graph Data Model and References

One of the most striking differences between Semantic Web data and XML is that XML does not allow multiple parent nodes for the same XML element and must be considered tree structured under this consideration. This is why most XML query languages such as XPath and XQuery provide a tree data model. Taking the special attributes `id` and `idref` into account, XML may also be viewed as a graph structure. When querying XML it may sometimes even be useful to consider these XML references as true parent-child relationships. In contrast, Semantic Web data is truly graph structured, in that predicates are the only way of specifying relationships amongst resources, and nodes of an RDF graph may very well have multiple incoming edges. RDF graphs are usually represented by triples without any explicit references. Nevertheless, a graph structure is implied by these triples, because RDF references are implicit in that they exploit the fact that RDF resources are uniquely identified by URIs.

Whereas for XML query languages, such as XPath, XQuery and XSLT a tree data model is a natural choice, versatile query languages that incorporate also Semantic Web data must adopt a graph data model.

From the beginning Xcerpt was designed to not only handle XML data, but also semi-structured graph data, which means that it can be adapted to natively handle Semantic Web data easier than other XML query languages.

2.2 Labeled Edges

Put simply, the XML data model is a node-labeled tree. In contrast, RDF graphs are not only node-labeled, but also edge-labeled. In XML serializations of RDF graphs such as RDF/XML, this difference is overcome by "striped" XML, which means that element nodes representing RDF nodes and edges alternate in the nested XML serialization. The *Syntactic Web Approach* suggests querying RDF

serializations with XML query languages. This solution is unsatisfactory in various ways: (1) It is not coherent with the visual and intuitive representation of RDF data as graphs, and is thus more difficult to grasp. (2) It does not pay tribute to the different roles assumed by subjects, objects and predicates of the RDF graph, which complicates e.g. the determination of the set of all predicates of an RDF graph. (3) Many XML serializations (such as RDF/XML and RDF/A) offer a great amount of variability and syntactic sugar for representing RDF graphs, which makes the formulation of queries against such serializations in XML query languages cumbersome.

As a result, a truly versatile query language for the Web must offer a data model that comprehends both: node- and edge-labeled graphs as well as purely node-labeled graphs. As has been mentioned before, node- and edge-labeled graphs can be transformed into graphs without edge labels in a straightforward manner. Nevertheless, the user must be provided with a syntax (see Section 3) that clearly distinguishes between edge- and node-labels both in query constructs and in the data.

2.3 Incomplete and Unbounded Data

In the Semantic Web, resources are uniquely identifiable, and thus anybody is free to make statements about resources by simply referencing the unique URI as subject, predicate or object within one's own statements. A consequence of this ability for everyone to make statements about arbitrary resources is that one may never be sure to be aware of all statements made about a given resource. From a graph perspective on Semantic Web data, this means that collecting all existing outgoing edges of a resource is not possible, which is a fundamental difference to XML data, where the sequence of children of an element node is fixed and can be determined simply by looking at the document containing the node in question.

To make things worse, whereas the size of the answers to XML queries is bounded by the document size, this does not hold for the size of the results of Semantic Web queries retrieving the information of interest for a single RDF resource, because the same resource may occur in multiple documents.

A possible solution (which also yields other benefits) to this problem is to restrict one's attention to the contents of specific documents or groups of statements, which are often referred to as *Named Graphs*. "Named graphs is the idea that having multiple RDF graphs in a single document/repository and naming them with URIs provides useful additional functionality built on top of the RDF Recommendations."[1] In fact, RDF query languages such as SPARQL and TriQL provide constructs for handling and constructing named graphs. An interesting issue to note is that named graphs provide a means to introduce completeness in RDF data.

The above observations show that the data model for a Semantic Web query language must be able to express both complete (in form of named graphs or doc-

[1] http://www.w3.org/2004/03/trix/

uments) and incomplete data (information that does not belong to any graph). While conventional Xcerpt query terms may already be complete and incomplete in breadth, data terms have always been considered to be complete. As shown in section 4 data terms can be naturally extended to include incomplete data, and an extended operational semantics that takes this extension of the data model into is being considered.

2.4 RDF Graphs as Xcerpt Data Terms

While in semi-structured data, there is always a distinguished top level term, the root, Semantic Web data does not have the concept of top level terms. Furthermore, it may not even be possible to single out a resource from which all other resources are reachable over edges in the graph, because RDF graphs may consist of disconnected subgraphs. It is, however, possible to determine a set resources, such that each resource in the graph is reachable from at least one of them. Choosing these resources as top level nodes, RDF graphs are very conveniently represented by sets of Xcerpt data terms.s

2.5 Order of Sub-Terms

Another difference between RDF and XML data illustrated in Figure 1 is that RDF data usually does not impose an order on outgoing edges of a node. To be more precise, RDF data is always unordered unless otherwise specified by the use of an `rdf:Seq` sequence container. Hence, the data model must be able to represent both ordered and unordered information. The distinction between ordered und unordered data is especially useful in the scope of positional queries against semi-structured data as exemplified in Section 4.

Xcerpt data terms have been conceived to not only represent XML data, but also semi-structured data in general. Therefore Xcerpt already supports the concept of unordered sets of children unlike most other XML query languages and does not need to be adapted to the Semantic Web in this respect.

Summing up the particularities of XML and RDF data, the data model must support possibly cyclic and disconnected graphs with labeled and unlabeled edges, complete and incomplete data specifications, ordered and unordered child elements, implicit and explicit references, and finally multiple roots.

3 An Intuitive Syntax for Versatile Web Query Languages

In previous work [5], we have shown that due to its versatility gained from construct-query-rules and constructs for treating heterogeneous data, Xcerpt is particularly well-suited to handle XML serializations for the Semantic Web data formats RDF and Topic Maps such as RDF/A, RDF/XML and XTM. An obvious alternative to processing XML serializations of Semantic Web formats is

their direct treatment. In fact, for Xcerpt's users it may be more convenient to use a syntax that better distinguishes between edges and nodes within an RDF graph. In this section, we propose a possible syntax derived from the syntax of Xcerpt data terms that represents RDF data in a very similar way to XML data.

Listing 1. The RDF Graph of Figure 1 represented as an Xcerpt data term

```
'montenegro@un.org'{
 <geo:bordersOn> _:country3{
   <geo:bordersOn> _:country1,
   <rdfs:label> literal('Albania')
 },
 <rdfs:label> literal('Montenegro')
}

_:country2 { <rdfs:label> 'USA' }
```

In listing 1 edges (predicates) of the RDF graph in figure 1 are enclosed by angle braces and appear in between the elements (subjects and objects) that stand for the nodes of the graph. This syntax eases the authoring and understanding of queries considerably, because subjects, predicates and objects are much more easily distinguished.

As has been mentioned above, data with labeled edges may be transformed to graph structured data with unlabeled edges by the introduction of an additional node for each edge. This approach has already been used to query Semantic Web data with Xcerpt in [11]. A graph data model with labeled edges can be offered to the user by the internal and automatic transformation of both RDF query and data graphs to graph data with unlabeled edges, which can already be handled by Xcerpt. In this article it is argued that the user of a versatile query language should be unconscious of and not be confronted with this transformation.

4 Common Query Constructs for the Web and the Semantic Web

Schema information often being unavailable, data on the Web is very heterogeneous. But even if schema information is present, it usually leaves room for variability. In contrast to relational database query languages, Web query languages must therefore provide constructs for handling this heterogeneity.

Besides querying Semantic Web data, programmers are also interested in transforming it. An example scenario for one such transformation is the collection of data from different sources, and its rearrangement according to a joint schema.

Xcerpt has been designed as a declarative language rooted in logic programming. This section shows that Xcerpt's approach to querying, transforming and reasoning is well-suited not only for ordinary semi-structured data, but also for the Semantic Web.

4.1 Query Patterns and Answer Closedness

One of the design principles of SPARQL and Xcerpt is answer closedness. This principle dictates that all answers to queries may themselves be used as queries. By ensuring similar syntaxes for both the formulation of queries and the representation of data, answer closedness eases program understanding.

Using data terms, it is just possible to check whether an RDF graph is entailed by the queried data, or whether a particular XML fragment is contained within a document. In order to extract parts of the data, queries must contain logical variables. Xcerpt query terms are data terms enriched by variables and a series of constructs for handling heterogeneous data. These constructs are just as useful in the Semantic Web as for ordinary XML data. Constructs for handling heterogeneity in Xcerpt include optional term selection, double braces for incompleteness in breadth and arbitrary length traversal path expressions.

One might be interested in all resources that represent countries directly or transitively bordering on Montenegro and their names. Assuming data of a similar form as in Figure 1, the following Xcerpt query in edge-labeled notation helps out:

Listing 2. An Xcerpt query term with constructs for handling heterogeneity

```
var Country →/.*/{{                                          1
  <rdfs:type> 'http://geo.org/country'{{ }},
  desc(<geo:bordersOn> /.*/)*                                3
    <geo:bordersOn> 'montenegro@un.org'{{ }},
  optional <rdfs:label> var Name →literal(/.*/)              5
}}
```

There are several noteworthy constructs in the above query term:

- *Variable Constraints*. In Line 1, the bindings for the variable `Country` is constrained to graphs matching the pattern following →.
- *Incompleteness in breadth*. The schema of data on the web is in many cases unknown. Therefore one might not know or even not care about the set of outgoing edges of an RDF node. Double curly braces are used in Xcerpt to indicate that the matched data may also contain additional siblings other than those specified by the query term.
- *Regular expressions* for labels. The logical variable `Country` in Listing 2 is supposed to be bound to all kinds of nodes within the queried RDF graph, no matter whether it is a blank node or a resource. The regular expression `/.*/` matches arbitrary URIs and b-nodes. In order to match just blank nodes or resources, the keywords `b-node` and `resource` can be used.
- *Incompleteness in depth*. The resource r_1 matching with the variable `Country` shall be directly or transitively connected over `geo:bordersOn`-predicates with the resource `montenegro@un.org`, which stands for Montenegro. The RDF nodes in between r_1 and `montenegro@un.org` are of no interest, and therefore an *arbitrary length traversal path expression* containing a wild-card regular expression for the resources of the intermediate nodes is used in line three.

– *Optional sub-terms.* Labels for the resources r_1 are to be retrieved if present. In the absence of such a label the query is not intended to fail, but to simply restitute no binding for the variable `Name`. Making use of the keyword `literal` ensures that `Name` is only bound to literals, never to URIs.

Solutions to Xcerpt queries are given in the form of substitution sets, which are sets of mappings from the logical variables in the query to subgraphs of the data. The query in Listing 2 applied to the RDF graph in Figure 1 yields the following substitution set:

{ {`Country` ↦ `_:country1`{ ... }, `Name` ↦ `'Albania'` } }

The fact that the variable `Country` is bound to the entire subgraph rooted at the resource it matches differentiates Xcerpt from other query languages such as SPARQL and RQL. Since in densely connected RDF graphs, the bindings of variables may contain large sub-graphs of the data or even the whole data graph, Xcerpt provides a second kind of variables called *label variables* which are not bound to entire subgraphs but only to the nodes they match with. The usage of a label variable in Listing 2 would be syntactically indicated by directly prefixing the double curly braces in Line 1 by the variable `var Country`.

Note that also SPARQL provides a way to return more information (entire subgraphs) about resources than just their URIs through the keyword `describe`. The exact nature of such descriptions is left unspecified by the SPARQL working draft, but the Concise Bounded Description[2] proposed by Nokia is mentioned as an example.

The semantics of the query in Listing 2 is implicitly defined by mapping the node-and-edge-labeled syntax of the query to purely node-labeled query terms.

4.2 Injectivity and Querying RDF Sequences

When specifying a query term to be matched with semi-structured data, the semantics intended by the query author is usually that sibling nodes shall *not* match with the same node of the queried graph. Listing 3 shows a query selecting all pairs of countries bordering on Montenegro, and Xcerpt's semantics[3] ensures that the variables `Country1` and `Country2` are not bound to the same node. Note that formulating a query that allows the bindings for `Country1` and `Country2` to be the same can be easily expressed using Xcerpt's `and` connective for queries.

Listing 3. A query selecting all pairs of countries bordering to Montenegro

```
montenegro@un.org{{
  <geo:bordersOn> var Country1 →/.*/{{ }},
  <geo:bordersOn> var Country2 →/.*/{{ }}
}}
```

[2] http://swdev.nokia.com/uriqa/CBD.html
[3] formally defined in [4, Chapter 8]

As has been mentioned in Section 2.5 Semantic Web data can both be ordered and unordered. Xcerpt's positional approach to querying allows to match data dependent on the order of sub-terms. Figure 2 contains a possible representation of information about spoken languages in countries using an RDF sequence container.

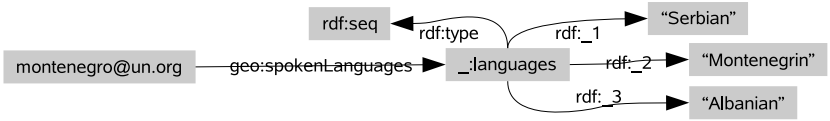

Fig. 2. An RDF sequence containing the languages in the order of their diffusion in Montenegro

The query in Listing 4.2 selects all countries in which Serbian is more common than Albanian assuming a schema as in Figure 2. The use of square brackets instead of curly braces indicates that the order of occurrence within the RDF sequence is relevant.

```
var Country →/.*/{{
 <geo:spokenLanguage> /.*/[[
  </.*/> literal('Serbian'),
  </.*/> literal('Albanian')
 ]]
}}
```

4.3 Blank Node Treatment

Blank nodes (also called b-nodes) in RDF graphs are used to assert that a resource r_1 exists that is related with other resources in a certain way without associating a URI to r_1. One unresolved issue related to querying RDF data containing b-nodes concerns the redundancy of answer sets. To see this reconsider the RDF graph from Figure 1.

Listing 4. A query selecting all resources of type `http://geo.org/country`

```
var Country →/.*/{{<rdfs:type> 'http://geo.org/country'}}
```

Selecting all resources of the graph that are of type `http://geo.org/country`, the query in Listing 4 cannot determine whether `_:country2` and `montenegro@un.org` are meant to be the same concepts. Hence, the question arises, whether both the blank node `_:country2` and the resource identified by `montenegro@un.org` should be returned or only the URI. The solution considered to be the most convincing by the authors is to exclude such query solutions that are entailed by other solutions, but to keep all others. The query in Listing 4

would therefore return both resources. In the case that the triple `(_:country2, rdfs:label, 'USA')` were not present, returning the blank node of the graph would be redundant.

4.4 Negation and Breadth-Complete Queries

As has been discussed in Section 2.3, Semantic Web data must be considered as inherently incomplete and unbounded in comparison to XML. Additionally taking into account that RDF statements are always positive assertions, the only sensible form of negation is scoped negation as failure, which has already been proven useful in the context of the Semantic Web[12, 13].

An approach that goes even beyond scoped negation as failure by providing explicit negative information to additionally enable strong negation is suggested in [14]. Although strong negation would certainly be helpful for Semantic Web Reasoning, it is not yet supported by Xcerpt.

While some Semantic Web query languages including the SPARQL family do not provide negation, XML query languages including Xcerpt usually do. To underline the importance of scoped negation in the Semantic Web consider the following query issued against the resource `http://countries.org/country_information`.

Listing 5. Scoped negation as failure in Xcerpt query terms

```
in{ resource{ 'http://countries.org/country_information' },
 /.*/{{
   <geo:bordersOn> 'montenegro@un.org'{{ }},
   <rdfs:label> var Name →literal(/.*/),
   not(<geo:bordersOn> /.*/{{ <rdfs:label>
       literal('Albania') }})
 }},
}
```

Listing 5 queries the names of all countries bordering to Montenegro but not to Albania. Matching a term with both positive and negated sub-terms with a data term is carried out as follows: At first, it is tested, whether each of the positive query sub-terms can be associated with a matching sub-term of the data respecting the injectivity requirement mentioned in Section 4.2. If this matching succeeds, it is searched for a matching sub-term of the data for the negated sub-terms. If any of the negated sub-terms can be matched, the entire matching fails. If all positive sub-terms can be matched with the data, but none of the negated ones, the entire matching succeeds. If all positive sub-terms can be matched with the data, but none of the negated ones, the entire matching succeeds. The query semantics for node- and edge-labeled query and data terms as needed for RDF data is ascribed to the semantics of purely node-labeled terms as described in [4, Section 8.2] by straight-forward normalization rules transforming edge-labeled terms to purely node-labeled terms.

Breadth-complete queries are an issue which is closely related to negated sub-terms, because they can be rewritten as breadth-incomplete queries using

the `without`-construct. They are indicated by single curly braces or brackets instead of double ones and can only be matched with data that does not contain any additional sub-terms besides those specified in the query term. To find countries that only border to Italy, the query in Listing 6 could be used.

Listing 6. Breadth-complete queries against RDF data

```
in{ resource{ 'http://countries.org/country_information' },
  var Country →/.*/{
    <geo:bordersOn> 'italy@un.org'{{ }},
  },
}
```

In the same way as queries with negated sub-terms, breadth-complete queries must be scoped to a single or a set of named graphs.

4.5 Optional Sub-Terms

As exemplified in Listing 2, optional constructs are of great help for Semantic Web queries in that they allow to extract certain parts of the queried data only if they are present. Closer examination of the `optional` construct reveals that it is only syntactic sugar for a disjunction of queries. The query in Listing 2 could also be written using the Xcerpt `or` construct:

Listing 7. The same query as in Listing 2 without the `optional` construct

```
or (
  var Country →/.*/{{
    <rdfs:type> 'http://geo.org/countries'{{ }},
    desc(<geo:bordersOn> /.*/)* <geo:bordersOn>
        'montenegro@un.org'{{ }},
    <rdfs:label> var Name →literal(/.*/)
  }},
  var Country →/.*/{{
    <rdfs:type> 'http://geo.org/countries'{{ }},
    desc(<geo:bordersOn> /.*/)* <geo:bordersOn>
        'montenegro@un.org'{{ }},
    without <rdfs:label> var Name →literal(/.*/)
  }}
)
```

As in SPARQL, multiple optional sub-terms may occur as siblings, or may even be nested. The semantics of such graph patterns seems to be straightforward at first glance: For each optional sub-term that succeeds to match, the bindings of its variables are included in the substitution set returned by the overall graph pattern. The failed matching of an optional sub-term does not prevent the overall graph pattern from returning a substitution set, which simply does not contain bindings for the variables in the unmatched optional sub-terms. Since variables may – and often do – occur multiple times in a query pattern, they may also be shared among multiple optional sub-terms, causing interdependencies among

them. In particular, it may happen that only one of two optional sub-terms may be matched, but not both. While the SPARQL working draft does not define which of the sub-terms is to be picked, Xcerpt adopts the following convention: If multiple optional sub-terms impede each other from matching, all selections of these sub-terms are chosen that maximize the number of variable bindings.

5 From Queries to Transformations

While most Semantic Web query languages are limited to querying and returning sets of mappings of their variables to resources, Xcerpt – and to some extent also SPARQL – are designed to do more: by providing construct terms (in SPARQL they are called graph templates) to be filled with the variable bindings gained from the evaluation of queries, they allow the construction of results having an entirely different schema. This combination of querying and construction in so-called *construct-query-rules* (see Section 5.1 for details) gives rise to the possibility of complex transformations.

5.1 Construct-Query-Rules and User Defined Reasoning

The evaluation of Xcerpt query terms and SPARQL graph patterns against RDF data yield substitution sets. Xcerpt construct terms are Xcerpt data terms enriched by variables as place holders and grouping constructs like `all` and `some`. Substitutions are applied to construct terms by replacing the variables in the construct term by their bindings in the substitution set (for the detailed semantics see [4, Section 7.3.3]). Query and construct terms are combined by so-called *construct-query-rules*, which allow sophisticated user-defined reasoning which goes beyond the predefined rules of RDFS and OWL.

5.2 Grouping Constructs

A major difference between SPARQL graph templates and Xcerpt construct terms is that only the latter allow merging of substitution sets (called result sets in SPARQL) by using grouping constructs. Merging substitution sets is necessary because often the need arises to collect variable bindings from different matches of the query pattern with the data. In contrast, a query result form within a SPARQL query is always filled exactly as often as the graph pattern in the `WHERE` clause matches with the queried RDF graph.

Reconsidering the information about countries and languages as exemplified in Figure 2, one might wish to construct an RDF graph that groups countries according to the languages which are spoken in them. To be more precise, for each language a blank node shall be constructed carrying an `rdfs:label` such as “Albanian”, “Serbian”, etc. Moreover the blank node must feature outgoing `geo:spokenIn` edges for each country that the language is spoken in.

Listing 8. Grouping countries according to languages

```
CONSTRUCT
 _:language{                                                    2
   <rdfs:label> var Language,
   all <geo:spokenIn> var Country,                              4
 }
FROM                                                            6
 var Country →/.*/{{
   <geo:spokenLanguage> /.*/{{ </.*/> var Language }}           8
 }}
END                                                             10
```

Using the grouping construct `all` (line 4), the query in Listing 8 collects all bindings for the variable `Country` that are contained within a substitution set for a fixed binding of variable `Language`. An important issue to note is that – just as in SPARQL – although the name `_:language` of the blank node in Line 1 is constant, a new blank node is constructed for each binding of the variable `Language`.

5.3 Versatile access to XML and RDF

Integrated access to different data formats includes the requirement that data should be easily transformed from one format to the other, and that different formats are queried simultaneously. As an exemplary use-case imagine that information about bordering countries is available in XML format structured similarly to that in the left part of Figure 1, and that information about languages spoken in these countries is only available in RDF format as in Figure 2.

The query in Listing 9 extracts all those pairs of border-countries whose citizens understand each other, because they speak the same language. The query part of the rule is a conjunction of two query terms, the first one querying the XML resource, and the second one drawing information from an RDF file. The names of countries sharing a common border are found by comparing the values of the `id` and `idref` attributes with a value join over the variable `ID` (in Xcerpt, XML attributes are enclosed in parentheses; double parentheses indicate that there may be additional unspecified attributes). Similarly, pairs of countries which have the same most common language are selected by a join over the variable `Language`.

Listing 9. Versatile access to Web data Formats in Xcerpt

```
CONSTRUCT
 result[                                                        2
   all understanding-neighbors[ var Name1, var Name2 ]
 ]                                                              4
FROM
 and (                                                          6
   in{ resource{ 'http://geo.org/Countries.xml' },
```

```
    Countries {{
      Country((var ID →id)){{ Name{ var Name1 } }},
      Country{{
        borderCountry((var ID →idref)),
        Name{ var Name2 }
      }}
    }}
   },
   in{ resource{ 'http://geo.org/languages.rdf' },
    /.*/{{
      <rdfs:label> var Name1,
      <geo:spokenLanguage> /.*/{{ <rdf:_1> var Language }}
    }},
    /.*/{{
      <rdfs:label> var Name2,
      <geo:spokenLanguage> /.*/{{ <rdf:_1> var Language }}
    }}
   }
  )
END
```

The query uses both constructs that are peculiar to either RDF or XML – such as variables for XML attribute values and edge-labeled query terms – and constructs that are applicable to both – such as complete and incomplete query term specifications. Notice that the variables `Name1` and `Name2` are shared among both conjuncts, which would be cumbersome to implement with two specialized languages for RDF and XML.

6 Conclusion and Outlook

Due to its graph data model, its rule-based nature and its convenient constructs for handling heterogeneity, Xcerpt turns out to be very well-suited not only for XML, but also for Semantic Web querying, transformations and reasoning. RDF data being increasingly made available as descriptive meta-data for HTML and XML documents, versatile access to both meta-data and XML in the same query program becomes ever more important for the next generation of web applications such as specialized search engines, and online booking and library systems. Developing such applications can be strongly eased by providing a query language that does not restrict itself to one of the formats, but provides integrated access to all of them, freeing the programmer from the burden of learning and combining multiple languages.

Besides laying the foundation for effective query authoring, a versatile query and reasoning language must process query programs efficiently in order to gain strong acceptance throughout the Web community. Several challenges are related to efficient query processing, demanding future work:

– Efficient parsing of semi-structured data from various serializations and efficient construction of in-memory graph representations of the data. Besides

parsing documents, in-memory graph representations must also be efficiently constructed from relational RDF stores.
- Efficient simulation unification of query patterns with graph data and construct terms. A large amount of research has been carried out in this direction concerning primarily tree queries, but also graph queries [6].
- Efficient backward chaining evaluation of programs. A forward chaining evaluation of Xcerpt programs is less reasonable because (a) the set of facts of an Xcerpt program can be very large, (b) the major part of derived facts may be irrelevant to the query (c) Xcerpt programs may have infinite fixpoints if they contain recursive rules.

Acknowledgements.

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. Boag, S., Chamberlin, D., Fernandez, M., Florescu, D., Robie, J., Simeon, J.: XQuery 1.0: An XML Query Language. W3C. (2005)
2. Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., Simeon, J.: XML Path Language (XPath) 2.0. W3C. (2005)
3. Clark, J.: XSL Transformations, Version 1.0. Recommendation, W3C (1999)
4. Schaffert, S.: Xcerpt: A Rule-Based Query and Transformation Language for the Web. Dissertation/Ph.D. thesis, University of Munich (2004)
5. Bry, F., Furche, T., Linse, B.: Let's Mix It: Versatile Access to Web Data in Xcerpt. Submitted for publication (2006)
6. Bry, F., Schroeder, A., Furche, T., Linse, B.: Efficient Evaluation of n-ary Queries over Trees and Graphs. Submitted for publication (2006)
7. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. Working draft, W3C (2006)
8. Karvounarakis, G., Magkanaraki, A., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M., Tolle, K.: Querying the Semantic Web with RQL. Computer Networks and ISDN Systems Journal **42** (2003) 617–640
9. Olson, M., Ogbuji, U.: Versa Specification. Online only (2003)
10. Bailey, J., Bry, F., Furche, T., Schaffert, S.: Web and Semantic Web Query Languages: A Survey. In Maluszinsky, J., Eisinger, N., eds.: Reasoning Web Summer School 2005. Number 3564 in LNCS. Springer-Verlag (2005)
11. Bolzer, O.: Towards Data-Integration on the Semantic Web: Querying RDF with Xcerpt. Diplomarbeit/Master thesis, University of Munich (2005)
12. Donini, F.M., Nardi, D., Rosati, R.: Description Logics of Minimal Knowledge and Negation as Failure. ACM Transactions on Computational Logic (2002) 177–225
13. Wagner, G.: Web Rules need Two Kinds of Negation. Principles and Practice of Semantic Web Reasoning (2003)
14. Analyti, A., Antoniou, G., Damasio, C.V., Wagner, G.: Stable Model Theory for Extended RDF Ontologies. International Semantic Web Conference 2005 (2005) 21–36