

# Xcerpt: Versatile Query Constructs and Data Model

François Bry, Tim Furche and Benedikt Linse

Institute for Informatics, University of Munich, Germany

June 11, 2006

# Motivation

- Semantic Web is gaining momentum
- Query languages are an accepted means for processing Semantic Web data: SPARQL, RQL, etc.
- Query languages are also an accepted means for processing XML data: XQuery, XSLT, Xcerpt
- Both Semantic Web data and XML applications are semi-structured.
- Common language constructs and a common data model can be identified.

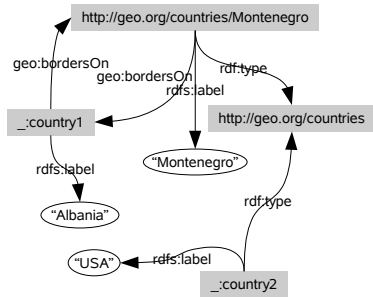
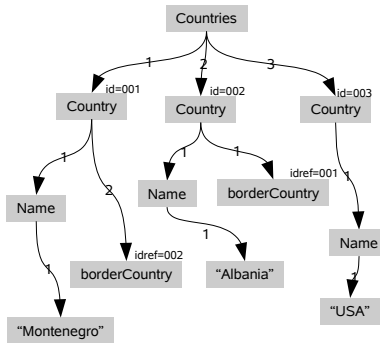
# Outline

- Motivation
- A quick introduction to Xcerpt
- Comparison of RDF and XML data
- RDF graphs as Xcerpt data terms
- Common Language Constructs for Querying
- Common Language Constructs for Construction and Transformations

# Quick Overview over Xcerpt

- Rule based query language
- Rich query patterns: *query terms* with logical variables
- Substitution sets: Sets of sets of variable bindings
- Construct terms: templates for constructing results
- Construct and query terms are connected by rules
- Evaluation of Rules with Backward Chaining

## XML versus RDF data



# Challenges for the Data Model

- Representation of graphs, not only trees
- Purely node-labeled graphs as well as node- and edge-labeled graphs
- Provision for XML-specificities (Attributes, id, idref, namespaces, entities, etc)
- Provision for RDF-specificities (Blank-Nodes, Literals, Containers and Collections)

# RDF Graphs as Xcerpt Data Terms

```
geo:Montenegro {  
  <geo:borders0n> _:country1{  
    <geo:borders0n> geo:Montenegro,  
    <rdfs:label> literal('Albania'),  
  }  
  <rdfs:label> literal('Montenegro'),  
  <rdf:type> geo:country,  
}  
  
_:country2 {  
  <rdfs:label> literal('USA'),  
  <rdf:type> geo:country  
}
```

# Common Language Constructs for Querying

Computing all countries that transitively border to Montenegro:

```
var Country -> /.*/{{
  <rdf:type> geo:country{{ }},
  desc (<geo:borders0n> /.*/)*
    <geo:borders0n> geo:Montenegro{{ }},
  optional <rdfs:label> var Name -> literal(/.*/)
}}
```

- Constructs for handling incompleteness
- Arbitrary length traversal path expressions
- Optional subterms
- Logical variables with structural restrictions

# Common Language Constructs for Querying

Computing all countries that transitively border to Montenegro:

```
var Country -> /.*/{{
  <rdf:type> geo:country{{ }},
  desc (<geo:borders0n> /.*/)*
    <geo:borders0n> geo:Montenegro{{ }},
  optional <rdfs:label> var Name -> literal(/.*/)
}}
```

- Constructs for handling incompleteness
- Arbitrary length traversal path expressions
- Optional subterms
- Logical variables with structural restrictions

# Common Language Constructs for Querying

Computing all countries that transitively border to Montenegro:

```
var Country -> /.*/{{
  <rdf:type> geo:country{{ }},
  desc (<geo:borders0n> /.*/)*
    <geo:borders0n> geo:Montenegro{{ }},
  optional <rdfs:label> var Name -> literal(/.*/)
}}
```

- Constructs for handling incompleteness
- Arbitrary length traversal path expressions
- Optional subterms
- Logical variables with structural restrictions

# Common Language Constructs for Querying

Computing all countries that transitively border to Montenegro:

```
var Country -> /.*/{{
  <rdf:type> geo:country{{ }},
  desc (<geo:borders0n> /.*/)*
    <geo:borders0n> geo:Montenegro{{ }},
  optional <rdfs:label> var Name -> literal(/.*/)
}}
```

- Constructs for handling incompleteness
- Arbitrary length traversal path expressions
- Optional subterms
- Logical variables with structural restrictions

# Common Language Constructs for Querying

Computing all countries that transitively border to Montenegro:

```
var Country -> /.*/{{
  <rdf:type> geo:country{{ }},
  desc (<geo:borders0n> /.*/)*
    <geo:borders0n> geo:Montenegro{{ }},
  optional <rdfs:label> var Name -> literal(/.*/)
}}
```

- Constructs for handling incompleteness
- Arbitrary length traversal path expressions
- Optional subterms
- Logical variables with structural restrictions

# The Semantics of the Optional Construct in Sparql and Xcerpt

```
_:x rdfs:label 'Germany'
_:x geo:borders0n Austria
_:x football:looses_sometimes_against England
```

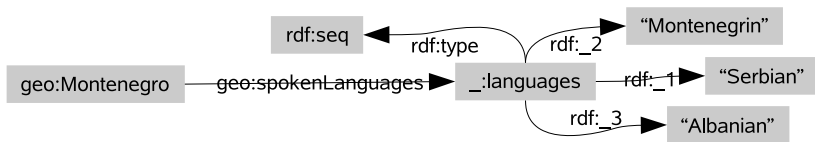
```
Var X -> /*./ {{
  <rdfs:label> literal('Germany'),
  optional <geo:borders0n> Y,
  optional <football:looses_sometimes_against> Y
}}
```

- in Sparql: one Substitution:  $Y \mapsto \text{Austria}$ ,  $X \mapsto \text{Germany}$
- in Xcerpt: two Substitutions: one for Austria, one for England.

# Other Xcerpt query constructs beneficial to the SW

- Breadth-complete queries
- Negation as Failure
- Injectivity constraints on siblings
- Order constraints on siblings (for RDF-Sequences)

# Ordered and Breadth-complete queries



```

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

# Grouping Constructs

- Most RDF query languages do not support sophisticated construction
- Deduction versus Transformation

CONSTRUCT

```
_:language{
  <rdfs:label> var Language,
  all <geo:spokenIn> var Country }
```

FROM

```
var Country -> /.*/{{{
  <geo:spokenLanguage> /.*/{{{ </.*> var Language }}} }}
```

END

# Versatile Access to Web Data with Xcerpt

CONSTRUCT

```
result[ all understanding-neighbors[ var Name1, var Name2 ] ]
```

FROM

```
and (
```

```
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
```

# Versatile Access to Web Data with Xcerpt

CONSTRUCT

```
  result[ all understanding-neighbors[ var Name1, var Name2 ] ]
```

FROM

```
  and (
```

```
    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
```

# Conclusion

- RDF and XML being complementary data formats makes integrated querying even more important.
- Xcerpt's data model, query constructs and rules are well suited for native RDF processing.
- Xcerpt's incompleteness specifications are powerful tools for querying also RDF data
- Access to both standard Web and Semantic Web data in a single query language is valuable and possible.

# Future Work

- Adjustment of Xcerpt's implementation to RDF
- Adaption of simulation unification to RDF graphs
- Incorporation of RDF/S and OWL reasoning capabilities
- Further optimization of Xcerpt's backward chaining algorithm

Thank You!  
Questions?