Informatics 1: Data & Analysis

Lecture 11: Navigating XML using XPath

Ian Stark

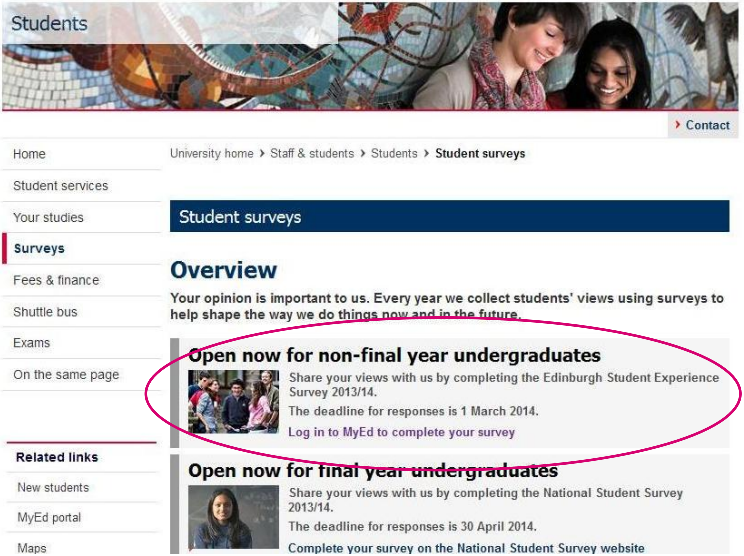
School of Informatics
The University of Edinburgh

Tuesday 25 February 2014
Semester 2 Week 6

Student Survey

- What: Edinburgh Student Experience Survey (ESES)
- Where: http://www.ed.ac.uk/students/surveys
- When: before 1 March 2014
- Why:
  - You will help influence what we do at Edinburgh, and through this your own future experience here
  - Generate a cash donation from the University to Edinburgh Student Charities Appeal / EUSA Academic Societies Fund
  - Entry into iPad prize draw

http://www.ed.ac.uk/students/surveys



XML

We start with technologies for modelling and querying *semistructured data*.

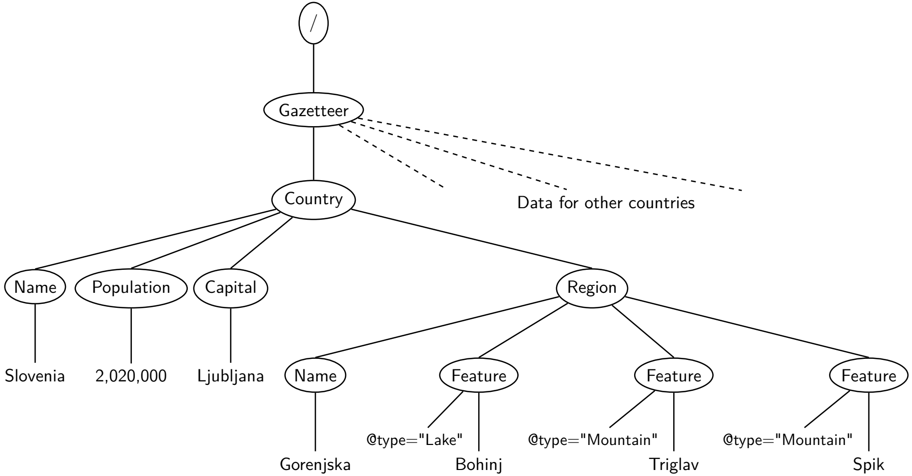
- Semistructured Data: Trees and XML
- Schemas for structuring XML
- Navigating and querying XML with XPath

Corpora

One particular kind of semistructured data is large bodies of written or spoken text: each one a *corpus*, plural *corpora*.

- Corpora: What they are and how to build them
- Applications: corpus analysis and data extraction

Sample Semistructured Data



Sample Semistructured Data in XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Gazetteer>
    <Country>
        <Name>Slovenia</Name>
        <Population>2,020,000</Population>
        <Capital>Ljubljana</Capital>
        <Region>
            <Name>Gorenjska</Name>
            <Feature type="Lake">Bohinj</Feature>
            <Feature type="Mountain">Triglav</Feature>
            <Feature type="Mountain">Spik</Feature>
        </Region>
    </Country>
    <!-- data for other countries here -->
</Gazetteer>
```

How to Extract Information from an XML Document?

Since an XML document is a text document, we could simply use conventional text search to look for data.

However, this ignores all the document structure.

A more powerful approach is to use a dedicated language for forming queries based on the tree structure of an XML document.

This is (yet another) *domain-specific language*.

With such a language we can, for example:

- Perform database-style queries on data published as XML;
- Extract annotated content from marked-up text documents;
- Identify information captured in the tree structure itself.

XQuery and XPath

XQuery is a powerful declarative query language for extracting information from XML documents.

As well as using XML documents for its source data, XQuery can also produce XML documents as output, so we can view it as an XML *transformation* language.

Interesting as the full XQuery language is, here we shall focus instead on a particular fragment.

XPath is a sublanguage of XQuery, used for navigating XML documents using *path expressions*.

XPath can be viewed as a query language in its own right.

It is also an important component of other XML application languages (XML Schema, XSLT, XForms, . . .).

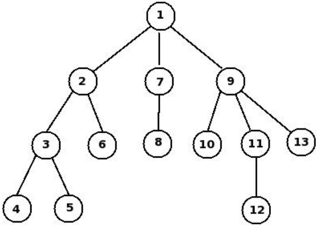
XPath Path Expressions

An XPath *path expression* (or *location path*) identifies a set of nodes within an XML document tree.

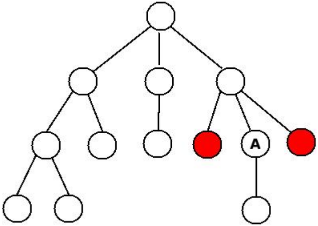
- The path expression describes a set of possible paths from the root of the tree.
- The set of nodes identified is all those reached as final destinations of these paths.

When using a path expression as a query on a document, this set of nodes is returned as a list (without duplicates) sorted in *document order* — the order the nodes appeared in the original XML document.

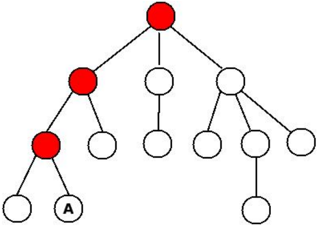
Family Tree Navigation



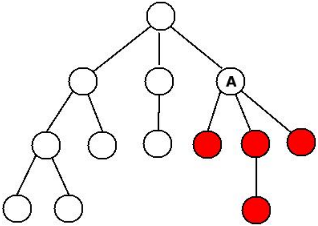
Document order



Siblings of A



Ancestors of A



Descendants of A

Examples of Path Expressions

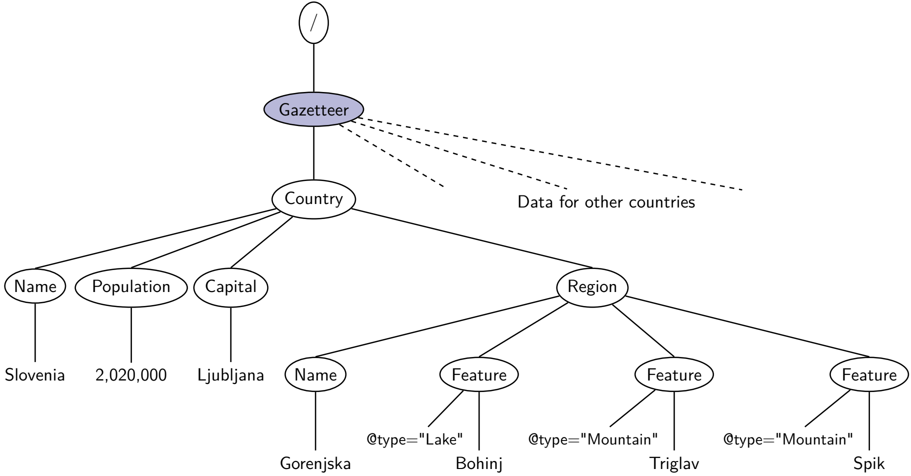
The next few slides illustrate a selection of path expressions applied to the gazetteer example. Each expression appears twice: once using a standard abbreviated syntax, and once using full XPath.

In each case, the nodes identified by the path are highlighted, and for a query would be retrieved in document order.

Paths are built up step-by-step as the path expression is read from left to right, with a *context node* that travels over the tree according to the components of the path expression.

The slash / at the start of a path expression indicates that the starting position for the context node is the document root.

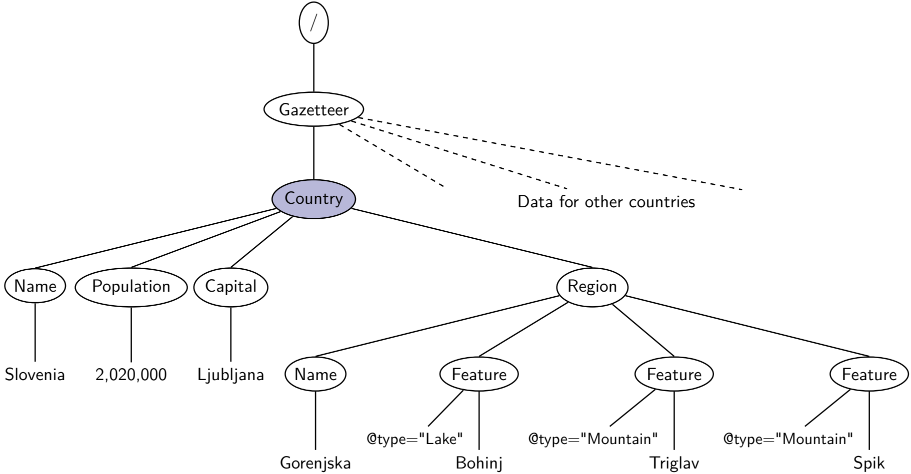
One Step



/Gazetteer

/**child**:: Gazetteer

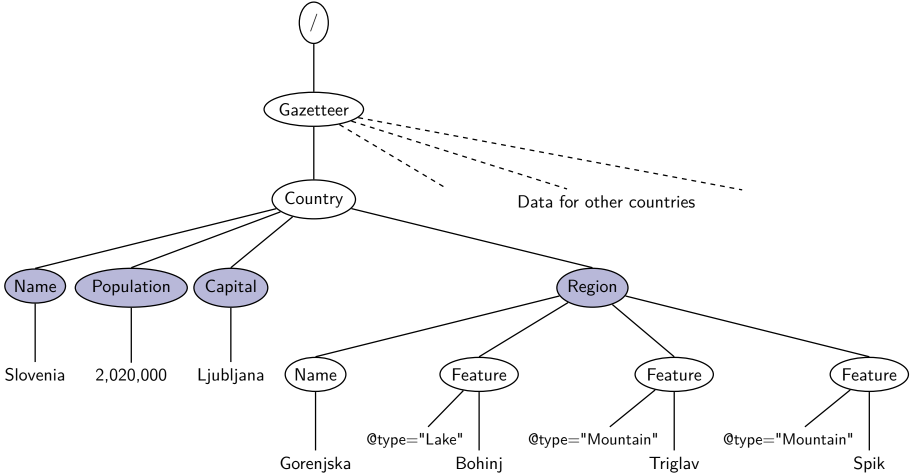
Two Steps



/Gazetteer/Country

/**child**::Gazetteer/**child**::Country

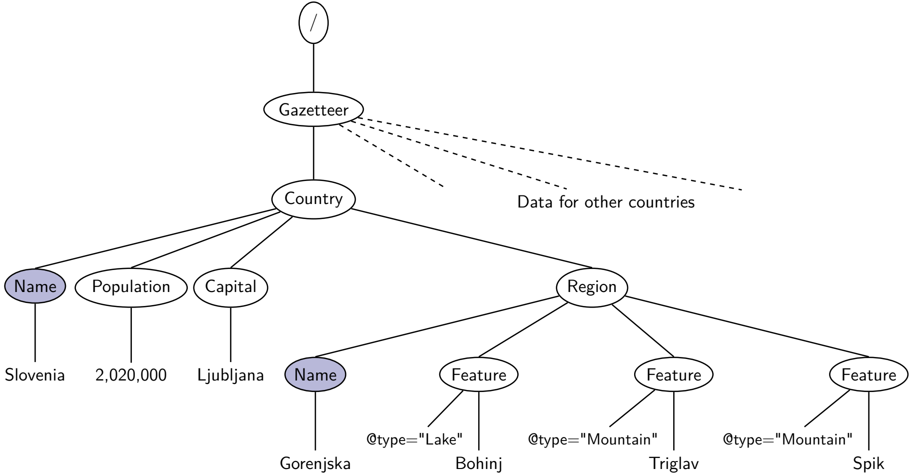
Children



/Gazetteer/Country/∗

/**child**::Gazetteer/**child**::Country/**child**::∗

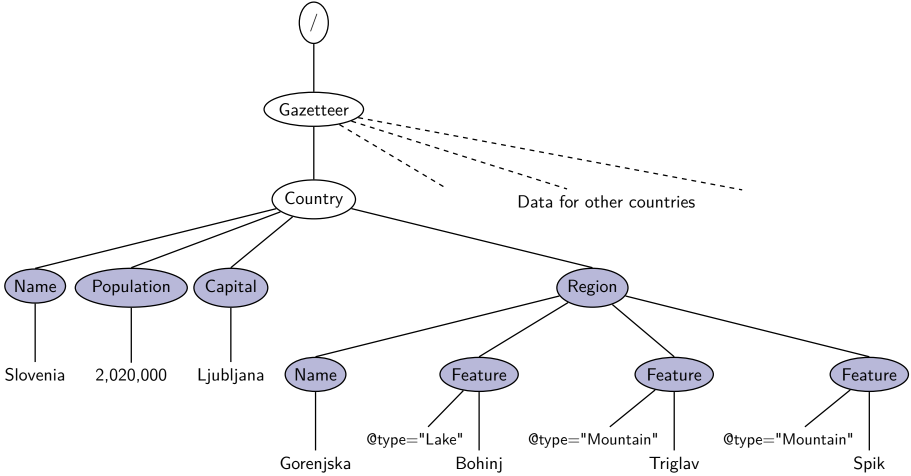
Many Steps



//Name

/**descendant**::Name

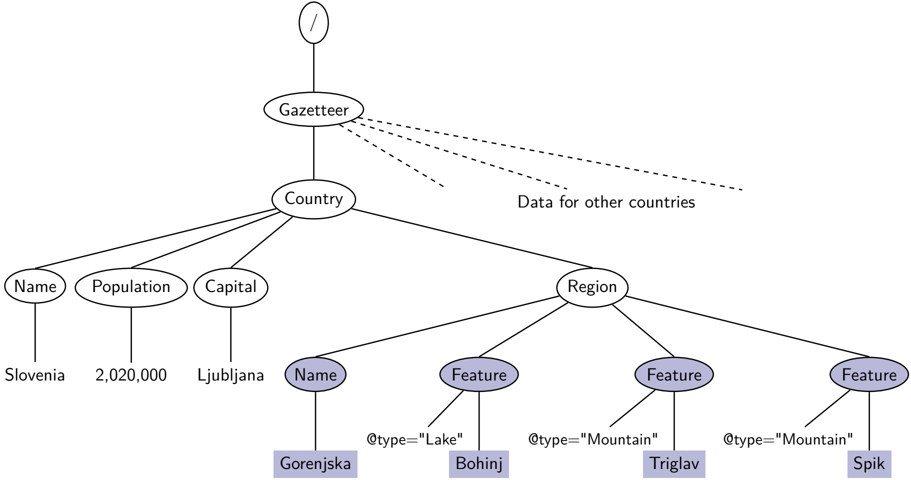
Matching Many Element Nodes



/Gazetteer/Country//*

/**child**::Gazetteer/**child**::Country/**descendant**::*

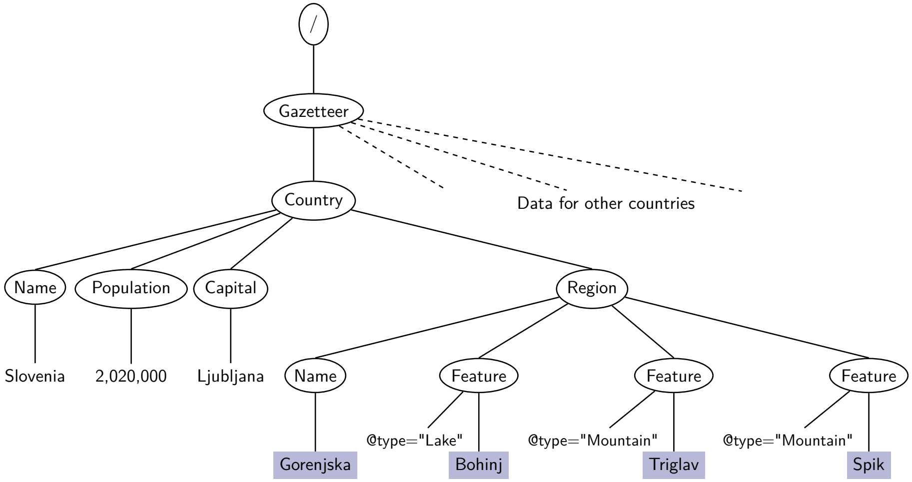
Matching Element and Text Nodes



//Region//node()

/**descendant**::Region/**descendant**::node()

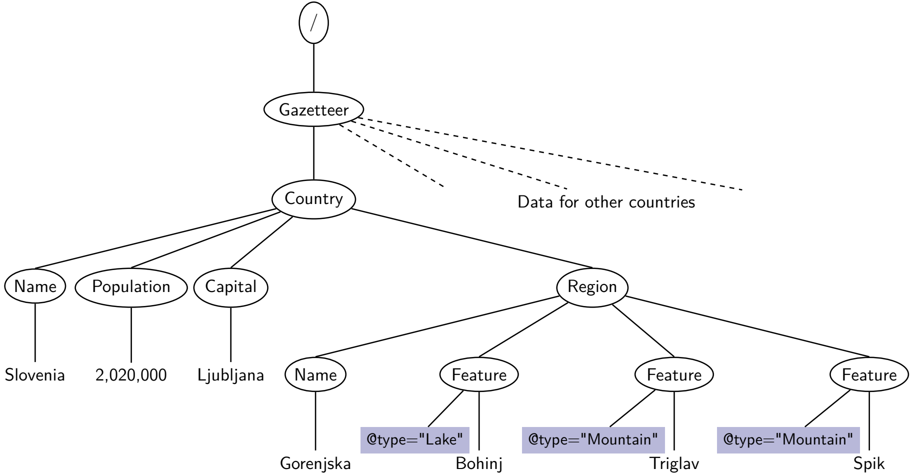
Matching Text Nodes



//Region//text()

//**descendant**::Region/**descendant**::text()

Matching Attribute Nodes



//Feature/@type

//**descendant**::Feature/**attribute**::type

Syntax for Path Expressions

A *path expression* is a sequence of *location steps* separated by a / character.

Each location step has the form

$$\langle \textit{axis} \rangle :: \langle \textit{node-test} \rangle \langle \textit{predicate} \rangle^*$$

- The *axis* indicates which way the context node moves.
- The *node test* selects nodes of an appropriate type.
- The optional *predicates* supply further conditions that need to be satisfied to continue with the path.

The examples so far used the child and descendant axes; node-tests node(), text(), $*$, and individual names; and no predicates.

Some Axes

Different axes point in different directions from the current context node.

- child: immediate children (attribute nodes don't count)
- descendant: any descendants (again, not attribute nodes)
- parent: the unique parent (root has no parent)
- attribute: all attribute nodes (context node must be an element node)
- self: the context node itself
- descendant-or-self: the context node together with its descendants.

Some Node Tests

Node tests select among all nodes along the current axis.

- text(): nodes with character data.
- node(): all kinds of node.
- $*$: all nodes of the "principal" node type for this axis: for the attribute axis, this is attribute nodes; for any other axis, element nodes. Never text nodes.
- *name*: element nodes with the given name.

  The names used for node tests in the earlier examples were: Gazetteer, Country, Region, Feature and type.

XPath Abbreviations

Complete path expressions can become cumbersome, and XPath provides a number of abbreviations for the basic operations.

- The child:: axis is default and can be omitted
- Syntax @ is an abbreviation for attribute::
- Syntax // is an abbreviation for /descendant-or-self::node()/
- Syntax .. is an abbreviation for parent::node()
- Syntax . is an abbreviation for self::node()

Some Predicates

The node test in a location step may be followed by zero, one or several predicates each given by an expression enclosed in square brackets.

[*locationPath*]

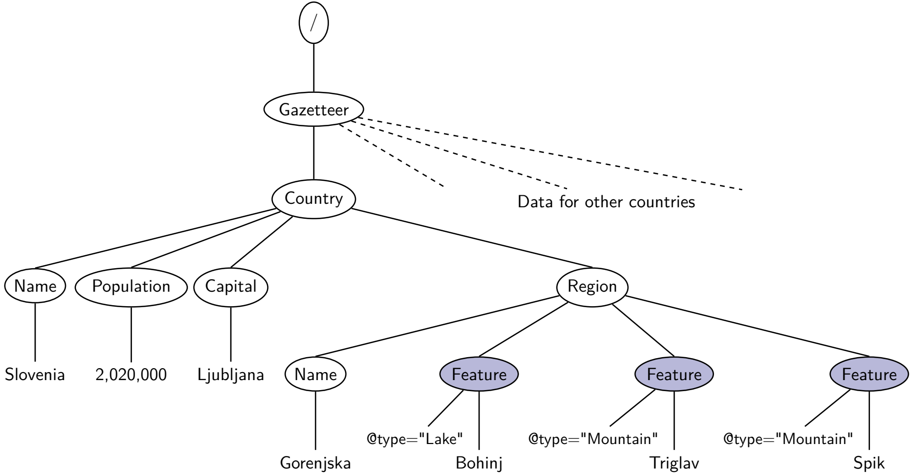
> Selects only those nodes for which there exists a continuation path matching *locationPath*.

[*locationPath=value*]

> Selects nodes for which there is a continuation path matching *locationPath* where the final node of the path is equal to *value*.

The full syntax of XPath predicate expressions includes arithmetic operations and further path queries, and is beyond the scope of this course.

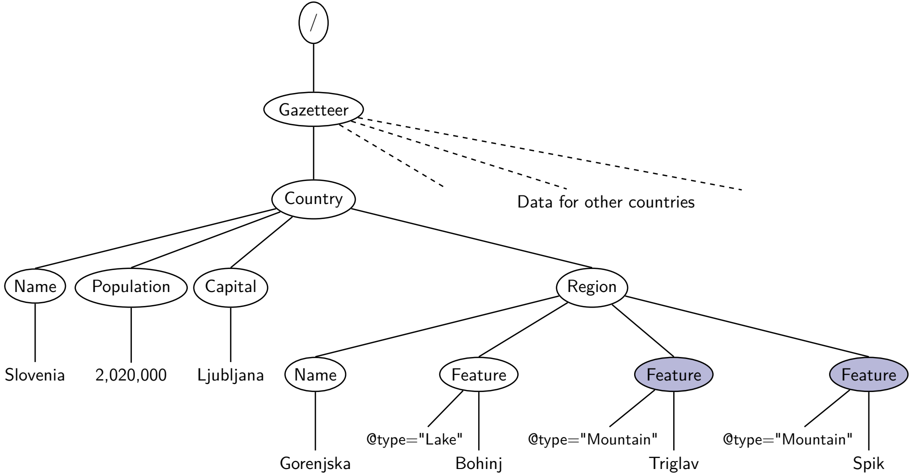
Path Predicate



//∗[@type]

/**descendant**::∗[**attribute**::type]

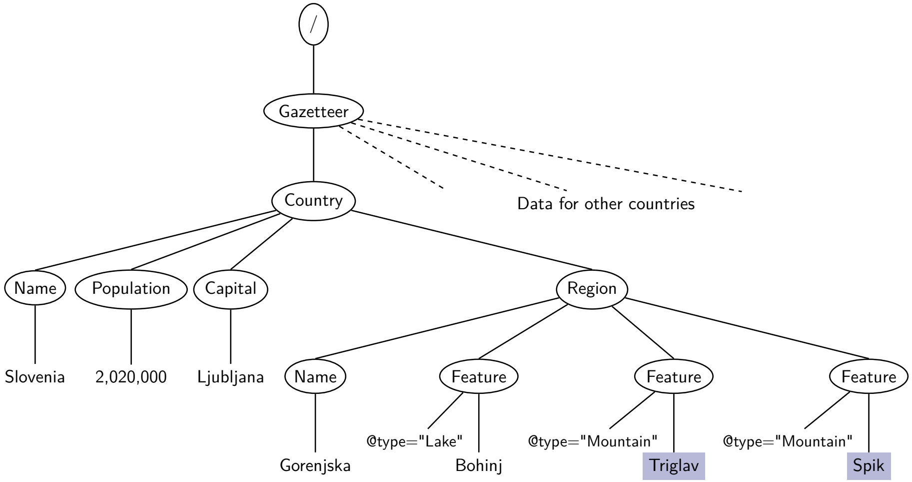
Path Predicate with Value



//*[@type="Mountain"]

/**descendant**::*[**attribute**::type="Mountain"]

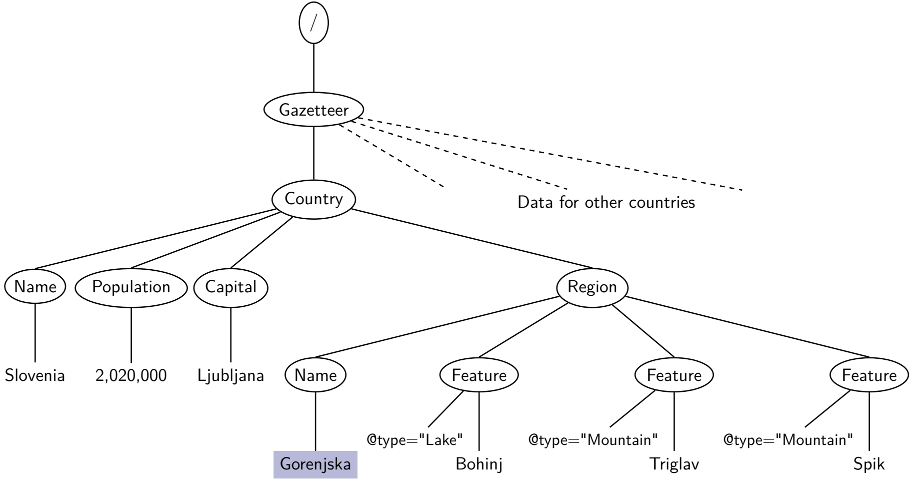
Path Predicate and Further Navigation



//∗[@type="Mountain"]/text()

/**descendant**::∗[**attribute**::type="Mountain"]/**child**::text()

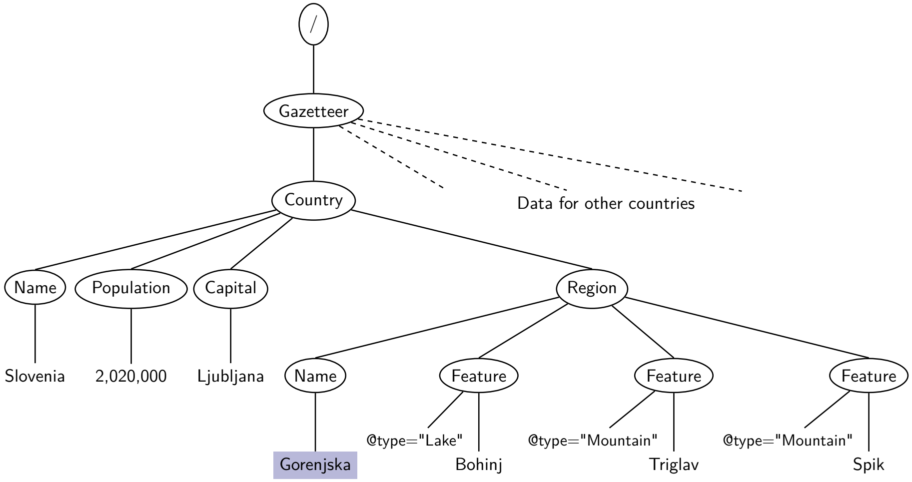
Navigation All Around



//Feature[@type="Mountain"]/../Name/text()

/**descendant**::Feature[**attribute**::type="Mountain"]/**parent**::∗/**child**::Name/**child**::text()

Different Ways to the Same End



//*[Feature/@type="Mountain"]/Name/text()

/**descendant**::*[Feature/**attribute**::type="Mountain"]/**child**::Name/**child**::text()

XPath as Query Language

These last examples begin to show XPath as a query language, in this case identifying in turn:

- All features which are mountains;
- The names of all mountains;
- The names of all regions containing mountains.

As with relational databases, a key challenge in implementing XPath and XQuery searches is not just to find algorithms that will do this, but to devise ones that will run efficiently on large XML datasets.

This is an active research area, with significant traffic from pure academic research to real-world impact.

Navigational Queries

XPath and XQuery use a *navigational* approach to formulating database queries. This was a standard model for database interrogation some decades ago, before the arrival of Codd's relational method.

Navigational querying, and its efficient implementation, has lately become a growing field — in part due to the rise in semistructured data and XML, but also the use of *graph databases* (remember Facebook Graph Search).

> A navigational query engine may have to do considerable work to transform an intuitive walk around a tree or graph into an appropriate form for efficient computation over large data.

Note on Paths to Descendants in Predicates

Name all countries containing a feature called "Salmon River"

We can select this from a gazetteer with the following XPath expression:

```
//Country[.//Feature/text()="Salmon River"]/Name/text()
```

Note the use of '.' to start a predicate path at the current context node.

However, this other — apparently very similar — expression won't do:

```
//Country[//Feature/text()="Salmon River"]/Name/text()
```

Without '.' the predicate //Feature/text() goes back to the root node.

More on XPath

Full XPath has a host of other features, including: navigation based on document order, position and size of context; name spaces; and a rich expression language. XPath 2.0 and XPath 3.0 add yet more.

Further Reading

The official W3C specification: http://www.w3.org/TR/xpath

Wikipedia on XPath: http://en.wikipedia.org/wiki/Xpath

The (wildly optimistic) *10-minute XPath Tutorial*: http://is.gd/xpath10

Homework

Tutorial sheet 5 will be online shortly. This involves writing an XML DTD and XPath queries, and testing them with the xmllint command-line tool.