ICAME Journal, Volume 39, 2015, DOI: 10.1515/icame-2015-0003

Modest XPath and XQuery for corpora: Exploiting deep XML annotation

Christoph Rühlemann, Philipps University, Marburg
Andrej Bagoutdinov, Ludwig-Maximilians University, Munich
Matthew Brook O'Donnell, University of Pennsylvania

Abstract

This paper outlines a modest approach to XPath and XQuery, tools allowing the navigation and exploitation of XML-encoded texts. The paper starts off from where Andrew Hardie's paper "Modest XML for corpora: Not a standard, but a suggestion" (Hardie 2014) left the reader, namely wondering how one's corpus can be usefully analyzed once its XML-encoding is finished, a question the paper did not address. Hardie argued persuasively that "there is a clear benefit to be had from a set of recommendations (not a standard) that outlines general best practices in the use of XML in corpora without going into any of the more technical aspects of XML or the full weight of TEI encoding" (Hardie 2014: 73). In a similar vein this paper argues that even a basic understanding of XPath and XQuery can bring great benefits to corpus linguists. To make this point, we present not only a modest introduction to basic structures underlying the XPath and XQuery syntax but demonstrate their analytical potential using Obama's 2009 Inaugural Address as a test bed. The speech was encoded in XML, automatically PoS-tagged and manually annotated on additional layers that target two rhetorical figures, anaphora and isocola. We refer to this resource as the Inaugural Rhetorical Corpus (IRC). Further, we created a companion website hosting not only the Inaugural Rhetorical Corpus, but also the Inaugural Training Corpus) (a training corpus in the form of an abbreviated version of the IRC to allow manual checks of query results) as well as an extensive list of tried and tested queries for use with either corpus. All of the queries presented in this paper are at beginners to lower-intermediate levels of XPath/XQuery expertise. Nonetheless, they yield fruitful results: they show how Obama uses the inclusive pronouns we and our as a discursive strategy to advance his political strategy to re-focus American politics on economic and domestic matters. Further, they demonstrate how sentence length contributes to the build-up of climactic ten-

sion. Finally, they suggest that Obama's signature rhetorical figure is the isocolon and that the overwhelming majority of isocola in the speech instantiate the crescens type, where the cola gradually increase in length over the sequence.

1 Introduction

The title of this article is reminiscent of the title of Hardie's paper "Modest XML for corpora: Not a standard, but a suggestion" (Hardie 2014). The similarity of the titles is not by chance. In this paper, we continue the line of thought Hardie initiated. He argued that for most corpus linguists and most purposes of corpus annotation a fairly limited set of suggestions for how to use XML are sufficient. One question, though, which arises immediately when you have completed annotating your corpus in XML format, was left unanswered: the question of how to exploit the XML annotation. This is no small question for a corpus linguist who has taken the (considerable) trouble to implement XML tags in his/her corpus: the tags are, after all, just a means to an end, together providing a vast resource of meta-information destined solely and expressly to be made use of in subsequent analyses. The question of how to mine a corpus by using this resource can therefore be considered central. In this paper, we address this question by introducing the XML Query Language, XQuery, a declarative programming language designed to function as a full query and scripting language for XML documents and databases, serving a role similar to that of SQL and stored procedures in relational databases. To be able to navigate XML documents and extract relevant information XQuery is dependent upon XPath. XPath provides a syntax to address elements in an XML document by specifying paths to walk through the 'tree' that represents the elements, attributes and text nodes it contains.

The aim in this paper is to advance the use of XPath and XQuery, which we have found to be immensely useful for corpus exploitation. To this end, we have set up an intertextual infrastructure around this paper. It consists of three related resources specifically designed to accompany the paper and which we have made available on a companion website. The infrastructure is depicted in Figure 1:

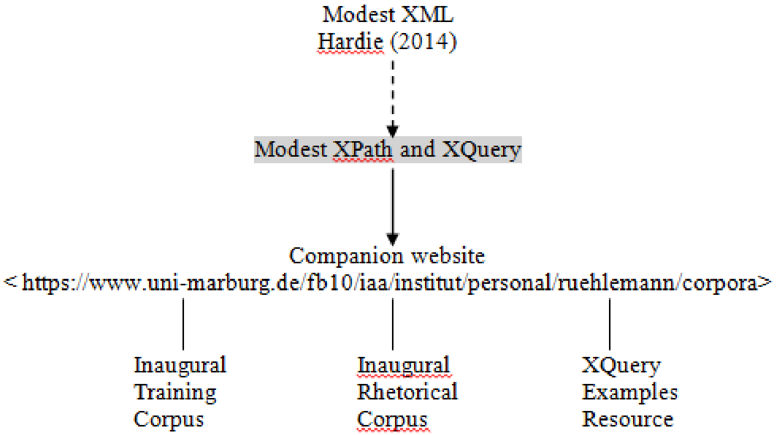

Modest XML
Hardie (2014)
Modest XPath and XQuery
Companion website
< https://www.uni-marburg.de/fb10/iaa/institut/personal/ruehlemann/corpora>
Inaugural Training Corpus
Inaugural Rhetorical Corpus
XQuery Examples Resource

Figure 1: Infrastructure of companion website

The three resources on the companion website are (i) the Inaugural Rhetorical Corpus (IRC), which will be described in more detail in Section 2, (ii) the Inaugural Training Corpus (ITC), a short extract from the IRC fully annotated to be used as a training corpus, and finally (iii) the XQuery Examples Resource, a file offering a large number of example queries (including advanced ones) for use with either corpus. We invite corpus linguists to make use of these resources.

XQuery is not yet widely used in corpus linguistics. The language is underlying the analyses in O'Donnell *et al.* (2012), an analysis of positioning in a newspaper corpus, O'Donnell and Römer (2012), a description of the annotation process used to encode MICUSP files, Rühlemann, O'Donnell and Bagoutdinov (2013), an examination of pauses in storytelling, and Rühlemann and O'Donnell (2015), a case study on introductory *this* in narrative. The only monograph we are aware of which is wholly based on the language is Rühlemann (2013),[1] a volume analyzing storytelling in conversation. The corpus underlying that study, the Narrative Corpus (NC; see Rühlemann and O'Donnell 2012), contains text samples extracted from the British National Corpus (BNC; see, for example, Hoffmann *et al.* 2008). It is worth looking at the structure of the annotation of that corpus in some more detail because it highlights why XQuery is useful for corpus analysis and why it may be even *more* useful than other analysis tools available.

The NC's annotation structure is 'deep' in the sense that it is characterized by complex hierarchy and multiple nesting of tags and can be contrasted with more 'shallow' annotation, characterized by fewer hierarchical levels and simpler forms of nesting. Compare (1), a very short excerpt from the BNC with (2), the corresponding excerpt from the NC. Given that the NC is derived from the BNC, the text contained in the <w>-elements is exactly the same. However, even a cursory look suffices to see that the hierarchy in the BNC excerpt is relatively simple, whereas the hierarchy in the NC excerpt is more complex:

(1) BNC:

```
<div n="009101" decls="KBERE000 KBESE000">
        <! -- <u>-elements -->
        <u who="PS04B">
                <! --  <s>-elements -->
                <s n="1944">
                        <mw c5="AV0">
→                         <w c5="AV0" hw="all" pos="ADV">All </w>
→                         <w c5="AV0" hw="right" pos="ADV">right </w>
                        </mw>
                        <w c5="PNP" hw="he" pos="PRON">he </w>
                        <w c5="VVD" hw="say" pos="VERB">said</w>
                        <c c5="PUN">.</c>
                        <! -- <w>-elements -->
                </s>
                <! --  <s>-elements -->
        </u>
        <! -- <u>-elements -->
</div>
```

In the BNC excerpt shown in (1), the hierarchy obtained for the two arrowed word elements *All* and *right* is five levels deep: the word elements nest in a multi-word (<mw>) element, which in turn nests within a sentence (<s>) element, which is embedded within an utterance (<u>) element, whose 'parent', finally, is a textual division (<div>) element. The contrast with the NC excerpt, shown in (2), is obvious. Here, additional levels of annotation apply such as sub-divisions for narratives in sequences of narratives (EC1, EC2, etc.) and textual components (e.g., CNN standing for 'story', CNI, standing for 'story-initial utterance') within these sub-divisions. As a consequence, the same two words, arrowed again, nest eight levels deep in the first <div> element; moreover, the two words are wrapped into an additional segment (<seg>) element, defining them as an instance of direct speech (tagged MDD). The full XPath for these two words (addressing just elements and not attributes) in this XML fragment is therefore: /div/div/seg/u/s/seg/mw/w.

(2) NC:

```
<div title="Fathers and daughters" embedLevel="EC">
  <div title="Women problems" embedLevel="EC1" narrativeType="T30">
    <seg Components="CNN">
      <seg Components="CNI">
          <! -- <u>-elements -->
          <u who="PS04B" Participation_roles="PNS">
                <! -- <s>-elements -->
                <s n="1944">
                  <seg Reporting_modes="MDD">
                      <mw c5="AV0">
→                       <w c5="AV0" hw="all" pos="ADV">All</w>
→                      <w c5="AV0" hw="right" pos="ADV">right</w>
                      </mw>
                  </seg>
                  <! -- <w>-elements -->
              </s>
              <! -- <s>-elements -->
          </u>
          <! -- <u>-elements -->
        </seg>
    </seg>
  </div>
</div>
```

The annotation hierarchy in excerpt (2) is by no means particularly deep; in the NC, hierarchies more than ten levels deep are not unusual. It is precisely this type of deep annotation project, with multiple nestings and complex hierarchies, that will benefit most from the use of XPath and XQuery. The reason is that many standard concordance and frequency tools provide little support for fully searching XML. Rather they "read files containing XML markup and either exclude the markup from the analysis or have primitive means of using a specific tag or tag combination to restrict processing (e.g. the Tags functionality in WordSmith Tools [Scott 2010])" (Rühlemann and O'Donnell 2012: 340-341).[2]

The ability to target with great precision any node(s) or node set(s) in an XML document using XPath and XQuery opens many possibilities for new types of questions that it would not be possible to address with standard corpus software (see Watt 2002 on XPath). Many tools, such as XML editors like oXygen and XML Spy, and programming languages, including R, Python, Perl, PHP, Java and C, provide XPath implementations (see Section 3 below) that make it possible to build scripts and tools with this potential. But here we argue particularly for the benefits of the XQuery programming language, which tightly integrates XPath into its syntax and provides a flexible yet simple framework for building queries for XML corpora.

In the following section (Section 2), we present the data selected for illustrative purposes, the Inaugural Rhetorical Corpus (IRC) containing President Obama's annotated 2009 Inaugural Address. In Section 3, we briefly describe environments in which XPath and XQuery operate. In Section 4, we focus on XQuery, describing its basic syntax and some major functions as well as exemplifying applications to the Inaugural Rhetorical Corpus. Section 5 concludes the paper.

2 Data: The Inaugural Rhetorical Corpus (IRC)

To highlight the potential of using XPath and XQuery, we selected Obama's first Inaugural Address (2009) as illustrative data. The speech was first fully PoS-tagged using the Free CLAWS WWW Tagger at Lancaster (http://ucrel.lancs.ac.uk/claws/trial.html), then converted into an XML document, and finally manually annotated on two additional levels, thus laying the foundations to the Inaugural Rhetorical Corpus (IRC), to which we hope to add in the future more annotated inaugural speeches. As noted, we have created a companion website where the IRC is available for download and use along with a training corpus (the Inaugural Training Corpus, ITC) and a number of queries designed specifically for the two corpora.

Why Obama's first Inaugural Address? It has been widely noted that Obama speeches are particularly rich in rhetorical figures (for example, Leith 2011). Rhetorical figures are discourse chunks whose spans can be anything between a single word and a sequence of sentences (or even more). Further, they need by no means act in series, one after another, but may get tied up within each other. Annotation designed to capture these (potentially intersecting) chunks will therefore be exactly of the kind of annotation that XPath and XQuery are best suited for, namely deep annotation.

The speech was coded in terms of what appeared to us the most salient rhetorical means, namely anaphora and isocola. In rhetoric, anaphora denotes, not backward-looking reference as in pragmatic studies, but the repetition of lexical material at sentence or clause beginnings. A prominent example of anaphora in political speeches is the repetition of "I have a dream" in Martin Luther King's 1963 March-on-Washington speech. Isocola are sequences of units ('cola') of the same ('iso') or a similar syntactic structure. The most oft-cited example is "veni, vidi, vici" attributed to Julius Cesar. In terms of positions in the sentence, isocola are less restricted than anaphora. While anaphora are by definition left-branching occurring in sentence- or clause-*initial* position, isocola can in principle, occur anywhere in the sentence or clause; generally though they tend to be right-branching occurring towards or at the end of sentences or clauses. The number of cola in isocola varies in the Obama speech between two and four: two same-structure elements count as dicola, three are referred to as tricola (for example, the above Cesar quote is a tricolon), and sequences of four symmetrical structures are called tetracola.

For either rhetorical figure we created an element of its own: <anaph> for anaphora and <isoco> for isocola. We further defined attributes and attribute values for the two elements. The attributes for <anaph> are 'number' (denoting the *n*-th time the anaphoric item is occurring in the sequence) and 'repeat' (denoting the word or words repeated). The three attributes for the isoco-elements include the following:

(i) 'ctype': the type of isocolon depending on the number of cola; values are 'di' for dicolon, 'tri' for tricolon, and 'tet' for tetracolon;

(ii) 'sequ': the number in the sequence; values are 'first', 'second', 'third'; 'fourth';

(iii) 'depend': the rank in the hierarchy of nesting <isoco> elements; values are 'indep' for isocola which are independent in the sense that they neither nest in another isocolon nor have any isocolon nesting in them; 'parent' for otherwise independent isocola which play host to a nesting isocolon; 'child' for isocola nesting within a parent-isocolon; and 'grandchild' for isocola nesting within a child-isocolon.[3]

Anaphora and isocola are illustrated in extract (3). The extract also highlights how the addition of merely two tags (for anaphora and isocola) can quickly create intricate interdependencies between them (and other elements) and thus 'deepen' the hierarchical structure considerably. For example, the tricolon of the '*all*-V' structure occurring at the end of the extract is child to the dicolon of the ‘*that*-N’ structure, which in turn is child to the tetracolon of the ‘*to*-V-N’ structure. The extract is given within its complete hierarchy, including the header and the highest-order tag <wtext> right above the text tags. (In the sentence preceding the extract, the anaphora “the time has come” has occurred for the first time; likewise, the first colon of a tetracolon has occurred there; hence the attribute values numb="n2" and sequ="second" for the anaphora and the isocolon appearing at the beginning of the extract.)[4]

(3)

```
<wtext type="OTHERPUB">
    <p>
    <! -- <p>-elements -->
        <s>
        <! -- <s>-elements -->
            <! -- <w>-elements -->
             <anaph numb="n2" repeat="the time has come">
                <w id="23.1" c5="AT0">The</w>
                <w id="23.2" c5="NN1">time</w>
                <w id="23.3" c5="VHZ">has</w>
                <w id="23.4" c5="VVN">come</w>
            </anaph>
            <isoco ctype="tet" sequ="second" depend="indep">
                <w id="23.5" c5="TO0">to</w>
                <w id="23.6" c5="VVI">reaffirm</w>
                <w id="23.7" c5="DPS">our</w>
                <w id="23.8" c5="AJ0">enduring</w>
                <w id="23.9" c5="NN1">spirit</w>
            </isoco>
            <c c5="PUN">;</c>
            <isoco ctype="tet" sequ="third" depend="indep">
                <w id="23.10" c5="TO0">to</w>
                <w id="23.11" c5="VVI">choose</w>
                <w id="23.12" c5="DPS">our</w>
                <w id="23.13" c5="AJC">better</w>
                <w id="23.14" c5="NN1">history</w>
            </isoco>
            <c c5="PUN">;</c>
            <isoco ctype="tet" sequ="fourth" depend="parent">
                <w id="23.15" c5="TO0">to</w>
```

```
<w id="23.16" c5="VVI">carry</w>
<w id="23.17" c5="AV0">forward</w>
<isoco ctype="di" sequ="first" depend="child">
  <w id="23.18" c5="DT0">that</w>
  <w id="23.19" c5="AJ0">precious</w>
  <w id="23.20" c5="NN1">gift</w>
</isoco>
<c c5="PUN">,</c>
<isoco ctype="di" sequ="second" depend="child">
  <w id="23.21" c5="DT0">that</w>
  <w id="23.22" c5="AJ0">noble</w>
  <w id="23.23" c5="NN1">idea</w>
  <c c5="PUN">,</c>
  <w id="23.24" c5="VVD">passed</w>
  <w id="23.25" c5="AVP">on</w>
  <w id="23.26" c5="PRP">from</w>
  <w id="23.27" c5="NN1">generation</w>
  <w id="23.28" c5="PRP">to</w>
  <w id="23.29" c5="NN1">generation</w>
  <c c5="PUN">:</c>
  <w id="23.30" c5="AT0">the</w>
  <w id="23.31" c5="AJ0">God-given</w>
  <w id="23.32" c5="NN1">promise</w>
  <w id="23.33" c5="CJT">that</w>
  <isoco ctype="tri" sequ="first" depend="grandchild">
    <w id="23.34" c5="DT0">all</w>
    <w id="23.35" c5="VBB">are</w>
    <w id="23.36" c5="AJ0">equal</w>
  </isoco>
  <c c5="PUN">,</c>
  <isoco sequ="second" ctype="tri" depend="grandchild">
    <w id="23.37" c5="DT0">all</w>
    <w id="23.38" c5="VBB">are</w>
    <w id="23.39" c5="AJ0">free</w>
  </isoco>
  <c c5="PUN">,</c>
  <w id="23.40" c5="CJC">and</w>
  <isoco ctype="tri" sequ="third" depend="grandchild">
    <w id="23.41" c5="DT0">all</w>
```

```
                    <w id="23.42" c5="VVB">deserve</w>
                    <w id="23.43" c5="AT0">a</w>
                    <w id="23.44" c5="NN1">chance</w>
                    <w id="23.45" c5="TO0">to</w>
                    <w id="23.46" c5="VVI">pursue</w>
                    <w id="23.47" c5="DPS">their</w>
                    <w id="23.48" c5="AJ0">full</w>
                    <w id="23.49" c5="NN1">measure</w>
                    <w id="23.50" c5="PRF">of</w>
                    <w id="23.51" c5="NN1">happiness</w>
                  </isoco>
                </isoco>
              </isoco>
              <c c5="PUN">.</c>
          </s>
          <! -- <s>-elements -->
    </p>
    <! -- <p>-elements -->
</wtext>
```

If we start counting with the highest-order element, <wtext>, which encompasses all other elements, the first anaphora is already three levels deep, while the last isocolon is embedded as many as six levels deep in the hierarchy of the XML structure. Keeping this structure in mind is crucial for building XPaths and developing XQuery scripts, as will be shown in the following sections.

3 *XPath and XQuery platforms*

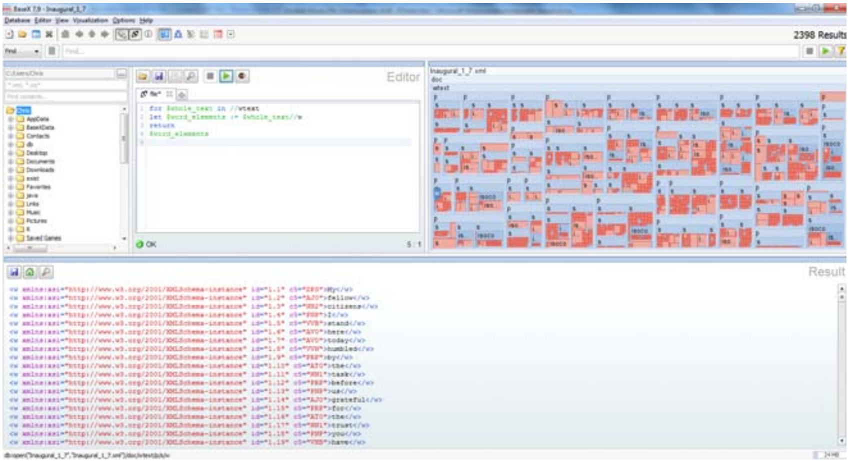
The XML Path Language, XPath (cf. Clark and DeRose 1999; Watt 2002), is a syntax and data model developed to address parts of an XML document. It models an XML document as a tree of nodes (elements, attributes and text). It might be helpful for some readers to consider the role of XPath within XML-based queries, and particularly in relation to XQuery, as analogous to that of regular expressions (or regexes) within text-based queries. Regular expressions allow for powerful matching of variable patterns and sequences of characters but require a ‘host’ environment or language to make use of them. Similarly, XPath queries can be constructed to match a particular element or set of elements in an XML document, but a host framework or language is required to further process the result, e.g. create a concordance listing or construct a frequency list.

Most XML-based tools, such as editors (see http://wiki.tei-c.org/index.php/Editors), provide XPath functionality allowing the user to carry out the equivalent of a search function which is aware of the XML document structure. Likewise, the majority of programming languages have either integrated or library support for XPath, among them, most notably, R (cf., for example, Gries 2009)[5] but also Python, Perl, PHP, Clojure, etc.

XQuery is an extension of XPath and shares "the same data model and the same set of built-in functions and operators" (Walmsley 2007: 13). Unlike XPath, XQuery is capable of processing multiple variables simultaneously and returning complex sets of diversified results (numerical, textual, and metainformational) in a single query. It is frequently implemented in environments with different architectures, operating systems and hardware, often in conjunction with related languages developed by the World Wide Web Consortium, e.g. XSLT.[6] XQuery itself is not a piece of software but a language. Just like natural languages require a medium for their articulation, so XQuery requires a medium, or platform to be articulated.

One platform for XQuery already tried and tested in corpus linguistic studies[7] is eXist-db, a Java-built database that can be installed by executing a Jaav package jar-file.[8] The path from installing eXist to actually performing queries is admittedly rather long-winded. During the installation process an administrator password must be set which will later be required for creating a collection of corpus files. After the installation and execution of the eXist-db Database launcher the so-called dashboard, a web-based GUI, can be opened by clicking the eXist icon in the taskbar and pressing 'Open dashboard'. This interface offers a large number of applications, of which 'Collections' and 'eXide' are most relevant for XQuery. Clicking on 'Collections' will first prompt the user to enter the username 'admin' and the password set during the installation. Using the 'Upload resources' button files can be imported from the user's hard drive. It is recommended to index the uploaded files by pressing the button 'Reindex collection' to reduce query processing time. Next, the integrated development environment, eXide, needs to be accessed (via the dashboard). Here, the target file is selected by clicking the button 'Open'. A click on 'New XQuery' opens the query window. To see whether a query works in the intended way, click on 'Eval'; this will output a limited amount of results. For queries intended to return limited output, such as frequencies, this function is perfectly sufficient. To get results for large-output queries, the query first needs to be saved (click on 'Save') before the 'Run' button will be enabled.[9] Finally, eXist facilitates the process of making corpora available to colleagues and comes with an active community and a large amount of available documentation (see Siegel and Retter 2014).

Another, 'lighter' and more user-friendly, solution is an open-source database called BaseX.[10] We recommend downloading the 'Core Package' as it is already fitted with a Java-based interface and does not require DTDs or other types of schemas to create collections (Mahlow *et al.* 2012: 201). The server application is cross-platform compatible and also requires the latest Java runtime environment to start. If required, BaseX allows multiple users to query or inspect XML corpora via a local network or the internet. After BaseX's execution, the user interface, shown in Figure 2, opens up immediately. To get started, users need to create a new database by opening the drop-down menu 'Database', selecting 'New' and choosing from their hard drive the XML file(s) they wish to work with. Queries can now be entered into the 'Editor' window; a click on the green arrow button will run them. The results are displayed in the 'Result' window in the lower half of the screen. Other helpful information, for example query information associated with the query, can be viewed by pressing 'Visualization'. In the upper right window BaseX offers a map of the queried XML file that highlights all nodes addressed by the query in color - a useful addition which may facilitate apprehension of XQuery and XPath syntax.[11] BaseX has already been successfully used by Mahlow *et al.* (2012) in the analysis of phraseological units in a diachronic German corpus. Mahlow paints a detailed picture of the possibilities BaseX offers and concludes that "[t]he capabilities of XQuery are, in our experience, powerful enough to effectively meet the requirements of corpus linguistics" (Mahlow *et al.* 2012: 203)[12].


BaseX 7.9 - Inaugural_1_7
Database Editor View Visualization Options Help
2398 Results
Find...
Editor
Result
OK
5 : 1
Inaugural_1_7.xml
doc
wtext

```
for $whole_text in //wtext
let $word_elements := $whole_text//w
return
$word_elements
```

```
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.1" c5="DPS">My</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.2" c5="AJ0">fellow</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.3" c5="NN2">citizens</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.4" c5="PNP">I</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.5" c5="VVB">stand</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.6" c5="AV0">here</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.7" c5="AV0">today</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.8" c5="VVN">humbled</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.9" c5="PRP">by</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.10" c5="AT0">the</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.11" c5="NN1">task</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.12" c5="PRP">before</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.13" c5="PNP">us</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.14" c5="AJ0">grateful</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.15" c5="PRP">for</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.16" c5="AT0">the</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.17" c5="NN1">trust</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.18" c5="PNP">you</w>
<w xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1.19" c5="VHB">have</w>
```

db:open("Inaugural_1_7","Inaugural_1_7.xml")/doc/wtext/p/s/w

Figure 2: Screenshot of BaseX user interface

4 *Using XQuery*

4.1 The most basic structures

XQuery boasts an extensive syntax. No attempt is made here to describe it in full detail. All that is possible is an outline of some basic syntactic structures and some useful applications. Three building blocks of XQuery's syntax stand out as essential: element structure, axis structure, and query structure.

4.1.1 Element structure

Elements are the main building blocks of XML. Their structure is tri-partite consisting of element, attribute, and attribute value. For example, //anaph selects all <anaph> elements (anaphora), //w selects all <w> elements (words). In XPath, not only elements but also attributes and attribute values can be addressed. Including attributes and values in the query serves to restrict the query to specific subsets of the elements. For instance, //isoco[@depend="child"] addresses only those isocola that have a 'child' value on the 'depend' attribute (that is, those isocola that nest in a higher-order isocolon) and //w[@c5="DPS"] selects only those words whose c5 attribute has the value 'DPS' (for possessive determiners, such as *my*, *our*, etc.). (Note that attribute values can be enclosed in either double or single quote marks; mixing the two forms in the same term is illegal.) One and the same element can be restricted by more than one restriction. Multiple restrictions are either coordinated by and within the same pair of square brackets or defined within separate brackets. For example, both //w[@c5='PNP'][ancestor::isoco] and //w[@c5='PNP' and ancestor::isoco] select personal pronouns occurring within isocola.

4.1.2 Axis structure

Key to using XQuery is firm knowledge of the hierarchical levels of the XML document in question. In XQuery, hierarchical levels are referred to as 'axes'. Axis labels are important orientation for the processor indicating "which 'direction' to head in as it navigates around the hierarchical tree of nodes" (Watt 2002: 82). To express the relations of superordination or subordination between the axes, a terminology is borrowed from human family relations. For example, the immediately next lower-order axis of the context node is referred to as child:: to that context node.

The most important axes are the following (for the full list, see for example Watt 2002; Walmsley 2007: 41-42):

self:: the context node itself (the term 'context node' is explained below)

parent:: the immediately next higher-order axis above of the context node

ancestor:: all higher-order axes above the context node (parents, grandparents, grandgrandparents, etc.)
child:: the immediately next lower-order axis below the context node
descendant:: all lower-order axes below the context node (children, grandchildren, grandgrandchildren, etc.)
following:: all nodes following the context node (minus descendants)
preceding:: all nodes preceding the context node (minus ancestors)

Crucially, axes are not assigned *a priori*, once and for all. Axis assignment always depends on, and varies with, the context node(s), that is, the element(s) from which you start your path. Suppose you have five elements <a> through <e>, of which <b> nests in <a>, <c> nests in <b>, d> nests in <c>, and <e> nests in <d>:

```
<a>
    <b>
        <c>
            <d>
                <e>
                </e>
            </d>
        </c>
    </b>
</a>
```

then the following axes obtain for the first three elements:

- <a> is parent:: to <b> and ancestor:: to <b> and all other elements
- <b> is child:: and descendant:: to <a>, parent:: to <c>, and ancestor:: to <c>, <d>, and <e>
- <c> is child:: to <b>, parent:: to <d>, ancestor:: to <d> and <e>, and descendant:: to <a>

A few example paths may illustrate how axes work:

//s[child::isoco] selects all sentences that host at least one isocolon (there are 76 of them; put the other way round, only 32 sentences, or less than a third, do not contain at least one isocolon). Conversely, //s[child::isoco/@depend='grandchild'] finds zero items because isocola whose 'depend' value is 'grandchild' are never direct children of <s> elements. To identify this type of sentence (<s> elements containing isocola whose 'depend' value is 'grandchild') the axis to be specified is the descendant:: axis: //s[descendant::isoco/@depend='grandchild'] (this path finds a single such sentence in the IRC).

To retrieve lexical patterns the following:: and preceding:: axes are useful. For example, //w[@c5='DPS' and following::w[1][starts-with(@c5,'NN')]] selects all possessive pronouns followed immediately (indicated by [1]) by a common noun no matter what number (singular or plural) (this is expressed by the function starts-with() and the omission of the number-specifying ‘1’ or ‘2’ from 'NN1' and 'NN2'; note that for this function the = sign conjoining attribute and value must be replaced by a comma).

4.1.3 Query structure

Queries of any sort and complexity are formulated as *clauses* following a basic structure, the so-called FLWOR structure (pronounced ‘flower’), with F standing for the for clause, L for the let clause, W for the where clause, O for the order clause, and R for the return clause. Given that our aim is to propose a *modest* approach to XQuery, we will not go into detail on where and order clauses, which are optional and much less often used, and instead focus on FLR only. The three clause types can be briefly characterized as follows.

To understand what the for clause does the above used metaphor of ‘navigating’ XML documents is quite helpful. Riding your car to your destination, you mentally trace a path from A_1 (your starting point) to B (your destination); if you start from another starting point, A_2, the path towards B will follow a different trajectory. The for clause defines your starting point–in XML parlance: it defines the context node or, if there is more than one, as is often the case, it defines the context nodes. Moreover, the for clause sets up a *loop* through these context nodes initiating an iterative process: each time a node in the corpus is found that matches the context node, a certain variable is bound to it (cf. Walmsley 2007: 7). While variables can, in principle, be labeled in any way, the label needs to be preceded by the dollar sign to be recognized as a variable label.

The for clause has the following structure: the components given in [ ] vary; all other components are part of the invariable syntax of the for clause:

```
for $[variable name] in //[context node(s)]
```

The function of let clauses is to associate a variable to a value; in most cases in a corpus analysis, this value will be the node(s) you wish to address – your ‘target nodes’.

The structure of a let clause is as follows:

```
let $[variable] := [value]
```

The return clause, finally, defines what variables you want to obtain; its simplest form is this:

```
return $[variable]
```

In the following section, we will show how these basic structures can be used to traverse and exploit the Inaugural Rhetorical Corpus. As is appropriate for an introduction, we will start from simple queries and gradually lead up to more complex ones.

4.2 Exploiting the Inaugural Rhetorical Corpus with XQuery

Perhaps the most common type of data corpus linguists work with are frequency lists (cf. Scott and Tribble 2006: Chapter 2). What is needed to compile such a list is an exhaustive inventory of all the word tokens occurring in a corpus. This inventory can be obtained easily using XQuery. As noted above, the first line of code, the for clause, requires the user to define a starting point, the context node(s). That is, the first question to be asked is which element(s) in the XML hierarchy are best suited as a starting point. There are several possibilities: the most direct path is via <w> elements; alternatively we can set out, for example, from the highest-order element <wtext>, or, one axis further down, from <p> elements, or yet another axis further down, from <s> elements. Going even deeper to the next axis, the <anaph> and <isoco> level, is not a good idea because not all <w> elements are wrapped into these elements. Note that if we choose <wtext>, the loop will stop after one iteration because there is merely one <wtext> element in the corpus; more loops would only be performed if we used as context nodes the <s> or <p> tags, of which there are many. The following two queries use //w and //wtext, respectively, as context nodes; they return exactly the same results:

```
for $word_elements in //w
return
$word_elements
```

```
for $whole_text in //wtext
let $word_elements := $whole_text//w
return
$word_elements
```

First three lines of output:

```
<w id="1.1" c5="DPS">My</w>
<w id="1.2" c5="AJ0">fellow</w>
```

```
<w id="1.3" c5="NN2">citizens</w>
```

As can be seen from the first three lines of the output, the query returns the words along with all the metainformation contained in the tag, such as 'id' and 'c5' attributes as well as the angle brackets around the tag. Obviously, the desired inventory contains just the words, stripped of all metainformation. To set up the query accordingly, we can re-use the above query with only a minor adjustment, namely the addition of the function string() to the return line. This will extract all the text contained within the matching element at any level, i.e. including nested elements:

```
for $words in //w
return
$words/string()
```

First three lines of output:

```
My
fellow
citizens
```

This query returns 2398 items; the list obtained looks much like the desired output. However, cursory inspection of the whole output (and the corpus) warns us to take this figure for real because CLAWS5 marks instances of genitive-*s* as words. To filter these ‘words’ out a restriction needs to be applied to the target item formulated in the let clause: instead of assigning the variable $words to each and every <w> element, as in the last query, the processor is told to look out for all word elements except those that have the value 'PUN' on the 'c5' attribute. That is, we modify the target node by using the function not(), passing as its argument the 'c5' value the genitive-*s* has:[13]

```
for $words in //w[not(@c5="POS")]
return
$words/string()
```

This query returns 2386 word tokens. Using appropriate software (for example, R) or XQuery,[14] the tokens can be counted, tabulated and processed further. Although hardly any corpus linguist needs to be persuaded of the value of frequency lists, it is worth taking a look at the list obtained for the Obama address, shown in Table 1:

Table 1: Top six most frequent words in the IRC

Rank	Word	Frequency	%
1	*the*	129	5.4
2	*and*	113	4.7
3	*of*	81	3.4
4	*to*	70	2.9
5	*our*	67	2.8
6	*we*	62	2.6

The fifth most frequent word, it turns out, is the possessive pronoun *our*, followed by the personal pronoun *we*. The high frequencies for either are unexpected considering the ranks they hold in frequency lists from large general corpora. For example, *our* is ranked 95th in COCA and 114th in the BNC, while *we* is ranked 30th and 41st respectively. The two words are clearly central in the address. Considering that *we* and *our* are deictics projecting a viewpoint (or 'origo') that is shared by speaker and audience,[15] the remarkable frequency of the two pronouns suggests a "discoursive strategy whose aim was to create identification and rapport between [Obama] and [his] audience" (Biria and Mohammadi 2012: 1294).[16] Given that Obama was primarily elected on his promise to bring the troops home from the wars in Iraq and Afghanistan, it would not be surprising if the centrality of *we* and *our* were indicative, not only of his discoursive strategy, but his *political* strategy to 'domesticize' America's political agenda after years of misguided wars.[17] We leave the task of testing this speculation to future research.[18]

So far, we haven't done much which other corpus tools cannot do (except, perhaps, for weeding out unwanted material such as the genitive *-s*); and obviously, there are simpler tools to generate a frequency list, for example antconc. So what is XQuery's sweet spot?

One set of questions that will bring us closer to an answer relates to the sentences in the corpus and the words contained in them. In a speech such as Obama's Inaugural Address, which is heavy with rhetorical means, we can expect that the length of sentences is not left to chance but will be used just as much for rhetorical purposes as almost anything else in the speech. To start, how many sentences are there? Again, this question is conveniently answered by defining <wtext> as the context node:

```
for $whole_doc in //wtext
let $n_sentences := count($whole_doc//s)
return
$n_sentences
```

This query matches all <s> elements in the document and returns the length of the resulting ‘nodeset’. There are 108 sentences in the speech. Considering that the number of words is 2386, the mean number of words per sentence is 22. This number is little exciting for we don’t know whether that is short or long and, after all, it’s just a mean. Means can hide important variation. So we will want to know how the sentences *vary* in terms of sentence length. In other words, how many words are there *per sentence*? To set up this query, it is advisable to select the <s> elements in the corpus as the context nodes that we start off from so the processor will loop over all <s> elements and count the number of <w> elements it finds in them. To instruct XQuery to *count* data the count() function is used (a crucial function for corpus linguists); here, we apply it to the target nodes. Also, as above, we wish to exclude the genitive -*s* from the count, so we re-use the appropriate restriction on the 'c5' attribute. This prompts the following query:

```
for $sentences in //s
let $words_per_s := count($sentences//w[not(@c5="POS")])
return
$words_per_s
```

First three lines of output:

```
28
23
8
```

Just as the above-mentioned mean, the first three lines of output conceal the amount of variation that is actually found in the data. Figure 3 plots the sentence lengths; the dotted black line across the figure represents the mean and the grey dashed line depicts a smoother, a locally weighted regression line:

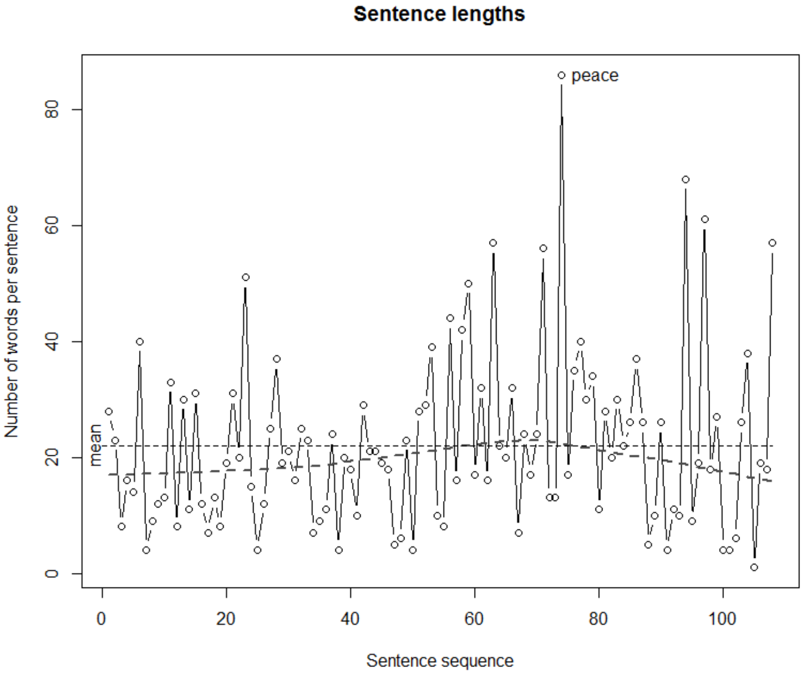


Figure 3: Sentence lengths (as measured by number of word tokens per sentence)

Inspection of the plot in Figure 3 suggests that sentence lengths vary a great deal, not only in terms of the wide span between the shortest sentence (1 word) and the longest sentence (86 words), but also in terms of rhythm: the speech seems to proceed in waves, as it were, alternating between short and long sentences. These waves build up noticeably towards the middle of the speech where, as a consequence, the smoother briefly rises above the mean whereas it hovers beneath it most of the time. At the very end of this build up we find the longest sentence in the speech, sentence number 74: its last word is the word *peace*:

(4) (…) America must play its role in ushering in a new era of peace.

Given the maximal length of the sentence and the gradual mounting of sentence lengths (while still maintaining the alternation between short and long sentences), the word *peace* is made particularly prominent. It appears that sentence length is used here as a climactic means to increase tension that is resolved in the climax – the appeal to America to *[usher] in a new era of peace*. Again, it is tempting to interpret the prominence awarded to *peace* in light of Obama's campaign promise to end the wars America was entangled in. (Obama's call for peace seems to have been heard in places as far from Washington as Oslo. Recall that just a few months after his speech Obama was awarded the Nobel Peace Prize!)

Above, we argued that XQuery is particularly suited to handle deep XML hierarchies. So far we haven't made that point clearly enough. For <s> elements are not particularly deep in the XML hierarchy of the IRC. What is, at times, very deeply embedded in the IRC's XML structure are the tags for the rhetorical figures anaphora and isocola as well as the words inside them.

Let us approach the rhetorical figures starting off with a seemingly simple question: How often does Obama use anaphora and isocola? This question is a two-fold one targeting two elements, <anaph> and <isoco>. For each target we could formulate an individual query. But, XQuery allows multiple nodes to be targeted at the same time. So we will set up a single query using two let clauses. Since we want output for two variables, we will use, in the return clause, the concat() function and pass the two variables as its arguments; as separators between the output for each variable, commas followed by straight quote marks can be used (note: the marks need to be separated either by white space, comma or semicolon; without any separation, the output will take the form of a contiguous string):

```
for $whole_doc in //wtext
let $n_anaphora := count($whole_doc//anaph)
let $n_isocola := count($whole_doc//isoco)
return
concat($n_anaphora,' ',$n_isocola)
```

The output is two numbers: 58 for <anaph> and 177 for <isoco> elements. However, these are not (yet) the numbers of anaphora and isocola in the IRC. Recall that in tagging the two rhetorical figures, the elements were *not* wrapped around the figure as a whole but around its components; for example, the <isoco> element encapsulates individual cola, not the isocolon as a whole.[19] So 177 is just the total number of cola in the corpus and 58 is just the number of times that lexical items were repeated at sentence beginnings. How to determine the numbers of occurrence of the figures as a whole?

Regarding anaphora what we are interested in, then, is the number of what we may call ‘anaphora units’, that is, groups of items belonging together through repetition. To establish the number of such anaphora units we make use of the <anaph> attributes 'numb' and 'repeat'. Since attributes and their values are contained in the element itself we can select the <anaph> elements as the context nodes and skip the let clause (in defining <anaph> as the context node we are addressing the element along with all the metainformation stored within it). To obtain the 'numb' and 'repeat' values we instruct the return clause to output these values for each <anaph> element using the string() function:

```
for $anaphora in //anaph
return
concat($anaphora[@numb]/string(@numb),';',$anaphora[@repeat]/
string(@repeat))
```

First five lines of output:

```
n1; so
n2; so
n1; our
n2; our
n3; our
n4; our
```

As can be seen from the first lines of the output, *so* (in its incarnation as predicative *so*, as in “*So it has been. So it must be …*”) is repeated once in two consecutive sentences, making up one anaphora unit, while the pronoun *our* is repeated thrice, making up the second anaphora unit in the speech. Going through the output in this way, we arrive at 21 anaphora units. Given the above noted keyness of *our* and *we* it may not surprise us that *we* and *our* figure very prominently in these units, *our* occurring altogether nine times in two anaphora units and *we* occurring 12 times in as many as four anaphora units.

To establish the number of isocola we need to find out how many incarnations there are of the three sequence-types of isocolon marked-up in the IRC (dicolon, tricolon, and tetracolon). To set up this count, we will again define <wtext> as the context node and assemble three let clauses, one for each sequence-type, using the 'ctype' attribute and its value ('di', 'tri', and 'tet', respectively) as a restriction:

```
for $whole_doc in //wtext
let $n_dicola := count($whole_doc//isoco[@ctype='di'])
let $n_tricola := count($whole_doc//isoco[@ctype='tri'])
let $n_tetracola := count($whole_doc//isoco[@ctype='tet'])
return
concat($n_dicola,' ',$n_tricola,' ',$n_tetracola)
```

The query returns three numbers: 106, 51, and 20. These however are, yet again, not the final numbers. The numbers need to be divided by the number of cola per sequence-type: there are, then, 106/2 = 53 dicola, 51/3 = 17 tricola, and 20/4 = 5 tetracola. As may be expected, frequency decreases with complexity. Adding up the three numbers, we arrive at 75 isocola in the IRC, clearly more than the number for anaphora, which was 21. Leith's (2011: 222) observation that anaphora is Obama's 'signature rhetorical figure' is not borne out by the 2009 Inaugural Address.

That Obama attaches greater importance to isocola than anaphora is also suggested by a look at the number of words occurring inside the figures. Counting the number of words inside anaphora is straightforward because anaphora do not nest in one another. The calculation of the word total for isocola is much less straightforward: as we have seen in the corpus extract in (3), isocola *do* nest in each other in very complex ways: one colon may be parent to a child colon, which in turn may be parent to another child (a descendant to the first parent colon). If this mutual nesting is not taken into account, the word count will return misleading values because words contained, for example, in a child colon would be counted twice: once for the parent colon and once for the child colon. To count the number of words contained in isocola more correctly two adjustments are needed. The first adjustment is that we will not use the double slash // but the single slash / in the let clause for isocola (line 3 in the code below). The double slash is an instruction to process all target elements (such as, in this case, <w> elements) at the descendant:: axis regardless of whether these targets represent or contain children, grandchildren, grandgrandchildren, and so forth (cf. Walmsley 2007: 45). The single slash, by contrast, is an instruction to process only targets at the child:: axis.

Restricting, then, the path to <w> elements that are immediate children to <isoco> elements causes a problem: the path will circumvent, and hence fail to count, those <w> elements that are enclosed in <anaph> elements, which are in turn enclosed in <isoco> elements, as exemplified in extract (5):

```
(5)
<s>
        <isoco ctype="di" sequ="first" depend="indep">
                <anaph numb="n1" repeat="so">
                        <w id="7.1" c5="AV0">So</w>
                </anaph>
                <w id="7.2" c5="PNP">it</w>
                <w id="7.3" c5="VHZ">has</w>
                <w id="7.4" c5="VBN">been</w>
                </isoco>
        <c c5="PUN">.</c>
</s>
```

Here, the word *so* is a child to the <anaph> element but a *descendant* to the superordinate <isoco> element – it will be overlooked by the single slash in line 3. To incorporate these descendant <w> elements in the word count for isocolon, the second adjustment is the addition of a third let clause (in line 4) to count the number of words of anaphora whose parent:: axis is an <isoco> element:

```
for $whole_doc in //wtext
let $n_w_anaphora := count($whole_doc//anaph//w[not(@c5="POS")])
let $n_w_isocola := count($whole_doc//isoco/w[not(@c5="POS")])
let $n_w_anaph.in.isoco := count($whole_doc//anaph[parent::isoco]//
w[not(@c5="POS")])
return
concat($n_w_anaphora,' ', $n_w_isocola,' ',$n_w_anaph.in.isoco)
```

The query returns three numbers: 102 (the total for <w> elements in anaphora), 1203 (the total for <w> elements that are children to <isoco> elements, and 45 (the total for <w> elements that are grandchildren to <isoco> elements). The grand total for isocola is thus 1248 words, which amounts to 52 percent of the 2384 words in the speech – an impressive number. The proportion for anaphora is merely 0.04 percent. Isocola can hence safely be considered the key rhetorical figure in the speech.

Let us now focus solely on isocola to address a slightly more complex question. As will be known, isocola can be of two types: *crescens* or *diminuens* (see Leith 2011: 277). An isocolon *crescens* is one in which the cola gain in length over the sequence, whereas an isocolon *diminuens* gradually decreases in length.[20] Both types of isocolon are climacto-telic (that is, 'aiming at a climactic increase or anti-climactic decrease'). The gradual increase or decrease in length can be seen as a schematic superstructure serving a rhetorical purpose: to inject into the isocolon what Longacker (1983: 4-6) called ‘tension’: a polarity gradually building up from the first colon, over potential intermediate cola, to the last colon. This climacto-telic tension exerts powerful effects on the audience: an isocolon *crescens* will bind the listeners’ attention, lend weight to what is expressed in the sentence, and, more importantly, tacitly evaluate it as desirable. (Conversely, an isocolon *diminuens* may tacitly depict its content as undesirable.)

How does Obama exploit the climacto-telic potential of isocola? We will approach this question by computing for each isocolon the number of words per colon; by comparing the numbers we will see in which direction they are headed – up (*crescens*) or down (*diminuens*).

It is important to understand that if the focus is to discover whether an isocolon is *crescens* or *diminuens* we need to include in the word count *all* words occurring in that individual isocolon regardless of whether some of the words are also contained in a child or descendant isocolon. Therefore, in defining the target nodes, we can use the double slash // again, instructing XQuery to count <w> elements of all <isoco> elements, irrespective of axes.

The following is the query for tricola. To address this sequence-type, the restriction [@ctype='tri'] is imposed on the context nodes in the for clause. The let clauses all restrict the $isocola variable to the familiar values on the 'sequ' attribute (see Section 2): 'first' for the first colon in the sequence, 'second' for the second colon, etc. Given the many nesting isocola, the order in which the word counts will appear in the output need not necessarily match the linear order of the 'sequ' values (for example, there may occur successive 'first' or successive 'second' values). To ascertain the correct match-up of 'sequ' value and word number the return clause is instructed to also output the 'sequ' value of each evaluated colon:[21]

```
for $isocola in //isoco[@ctype='tri']
let $words_first.colon := count($isocola[@sequ='first']//w[not(@c5="POS")])
let $words_second.colon := count($isocola[@sequ='second']//
w[not(@c5="POS")])
let $words_third.colon := count($isocola[@sequ='third']//w[not(@c5="POS")])
let $sequ_value := $isocola[@sequ]
return
concat($isocola[@sequ]/
string(@sequ),';',$words_first.colon,';',$words_second.colon,';',$words_third.co
lon)
```

First three lines of output:

```
first; 4; 0; 0
second; 0; 2; 0
third; 0; 0; 2
```

The first three lines of the results give the lengths of three cola. The three 'sequ' value strings on the left clearly indicate that all three belong to the same tricolon (this is also indicated by the zeros which are due to the fact that the three variables in the let clauses were defined for every <isoco> element; the variables are therefore evaluated for every colon in every isocolon). As can be seen, this first tricolon is *diminuens*: the length of the initial colon is four words, while the middle and the final one are two words long. To what extent is this first result representative of the results as a whole?

As shown in Table 2, the result is *not* representative: the clear majority (53.3 percent) of isocola is of the *crescens* type; only 17.3 percent are of the *diminuens* type. To qualify as *crescens*, the increase in number of words over the sequence of cola had to be uninterrupted. That is, a sequence of the structure [first=second<third], as in 1-1-2, or [first<second<third], as in 1-2-3, was counted as *crescens*. Conversely, [first=second>third], as in 2-2-1, and [first>second>third], as in 3-2-1, were treated as *diminuens*. Finally, [first=second=third], as in 1-1-1, was labeled ‘level’ while [first>second< third], as in 2-1-3, was labeled ‘wave’.

Table 2: Distribution of climactic isocolon types

Type	**Frequ**	**%**
crescens	40	53.3
diminuens	13	17.3
level	15	20.0
wave	7	9.3
Total	75	100.0

For illustration, let us return to *peace*: this word is not only, as noted, the last word in the longest sentence in the speech but also the concluding word in a tetracolon *crescens*, the sequence of number of words per colon being 7-8-12-14:

(6)

```
<s>
        <! -- <w>-elements -->
        <w id="74.39" c5="PNP">we</w>
        <w id="74.40" c5="VM0">can</w>
        <w id="74.41" c5="XX0">not</w>
        <w id="74.42" c5="VVI">help</w>
        <w id="74.43" c5="CJC">but</w>
        <w id="74.44" c5="VVB">believe</w>
        <isoco ctype="tet" sequ="first" depend="indep">
                <w id="74.45" c5="CJT">that</w>
                <w id="74.46" c5="AT0">the</w>
                <w id="74.47" c5="AJ0">old</w>
                <w id="74.48" c5="NN2">hatreds</w>
                <w id="74.49" c5="VM0">shall</w>
                <w id="74.50" c5="AV0">someday</w>
                <w id="74.51" c5="VVI">pass</w>
                </isoco>
                <c c5="PUN">;</c>
        <isoco ctype="tet" sequ="second" depend="indep">
                <w id="74.52" c5="CJT">that</w>
                <w id="74.53" c5="AT0">the</w>
                <w id="74.54" c5="NN2">lines</w>
                <w id="74.55" c5="PRF">of</w>
                <w id="74.56" c5="NN1">tribe</w>
                <w id="74.57" c5="VM0">shall</w>
                <w id="74.58" c5="AV0">soon</w>
                <w id="74.59" c5="VVI">dissolve</w>
        </isoco>
        <c c5="PUN">;</c>
        <isoco ctype="tet" sequ="third" depend="indep">
                <w id="74.60" c5="DT0">that</w>
                <w id="74.61" c5="PRP">as</w>
                <w id="74.62" c5="AT0">the</w>
                <w id="74.63" c5="NN1">world</w>
                <w id="74.64" c5="VVZ">grows</w>
                <w id="74.65" c5="AJC">smaller</w>
                <c c5="PUN">,</c>
                <w id="74.66" c5="DPS">our</w>
                <w id="74.67" c5="AJ0">common</w>
                <w id="74.68" c5="NN1">humanity</w>
                <w id="74.69" c5="VM0">shall</w>
                <w id="74.70" c5="VVI">reveal</w>
                <w id="74.71" c5="PNX">itself</w>
            </isoco>
            <c c5="PUN">;</c>
             <w id="74.72" c5="CJC">and</w>
            <isoco ctype="tet" sequ="fourth" depend="indep">
                <w id="74.73" c5="CJT">that</w>
                <w id="74.74" c5="NP0">America</w>
                <w id="74.75" c5="VM0">must</w>
                <w id="74.76" c5="VVI">play</w>
                <w id="74.77" c5="DPS">its</w>
                <w id="74.78" c5="NN1">role</w>
                <w id="74.79" c5="PRP">in</w>
                <w id="74.80" c5="VVG">ushering</w>
                <w id="74.81" c5="PRP">in</w>
                <w id="74.82" c5="AT0">a</w>
                <w id="74.83" c5="AJ0">new</w>
                <w id="74.84" c5="NN1">era</w>
                <w id="74.85" c5="PRF">of</w>
                <w id="74.86" c5="NN1">peace</w>
            </isoco>
            <c c5="PUN">.</c>
    </s>
```

In other words, *peace* is indeed *the* key word in the whole speech: it is not only the end point of the longest sentence, which is itself already noteworthy given the gradual increase in the length of the sentences before it, but it is also the culmination point of a rising tetracolon, the most complex rhetorical structure in the text. It would be hard to overstate the rhetorical effort Obama expends in winning his fellow citizens over for his course towards this goal.

5 Concluding remarks

In this paper we have presented a modest approach to XPath and XQuery. No claim was made to describe the syntax of the languages exhaustively; our treatment was instead explicitly selective, its selectiveness dictated by space considerations, which prevent discussion of more queries, and the specific XML structure of the IRC, which allows only a limited set of questions to be asked. The syntax covered is therefore decidedly basic. If readers feel persuaded to experiment with XPath and XQuery for their corpus project they are welcome to make use of the additional resources made available on the companion website. Even then, however, they will undoubtedly discover that this initial treatment and the companion resources are at best a beginning and that in order to answer the questions arising from *their* project they need to get immersed in the two languages beyond what we can offer. So the question is legitimate why corpus linguists should bother to develop XPath and XQuery skills. We would like to approach this question from three angles.

First, mastery of an initial level of XPath and XQuery syntax can be sufficient to extract instructive data and provide worthwhile insights. We hope to have made this point using the Inaugural Rhetorical Corpus, where the illustrative analyses did shed light on discoursal and rhetorical structure otherwise hidden to the naked eye. For example, we have been able to demonstrate the centrality of peace in the address, using parameters such as sentence lengths and climactic type of isocola.

Second, acquiring XPath and XQuery skills might be considered worth the effort now that there is a growing awareness in the corpus-linguistic community of the necessity and usefulness of acquiring such programming skills. As Gries (2010: 123) notes with regard to R, another programming language (and statistical environment) (see Gries 2013):

> And why is it that we as corpus linguists often must retrieve complex patterns from gigabytes of messy data in various encodings and forms of organization, but most of our curricula do not contain even a single course on basic programming skills or relational databases (while psychologists, computational linguists, cognitive scientists etc. devote years to acquiring the required methodological skills)? (Gries 2010: 123)

Given the growing complexity of corpus data programming seems inevitable in the long run. While, in an ideal world, a single programming language would “allow you to perform just about *all* corpus-linguistic tasks with one programming environment” (Gries 2009: 3), it is as yet an open question whether in the real world this all-purpose programming environment can indeed be found. It is probably not found in XQuery, which is tailor-made to suit XML-encoded data and cannot handle many other data types[22] and which is certainly less convenient than R for performing statistical evaluation tasks.[23] But with XML data it fulfills its purpose impeccably and elegantly, making it the natural choice for exploitation of this data type. (And it works smoothly in combination with R, the language in which, for example, the the analytical diagrams in this paper were programmed.)

Third, XQuery is the natural accoutrement to go with deep annotation capturing phenomena far above the surface level. PoS-tagging may be advanced enough for lexical and lexico-grammatical studies but its usefulness for discourse or pragmatic analyses is certainly limited (cf. Leech 2007: 134). Once annotation takes aim at discourse or pragmatic phenomena this will inevitably lead to more complex XML hierarchies. Deep annotation will in most cases require manual implementation, which will in turn limit the size of the corpus. Limited size need not be a downside. It can be made up for by specificity (specific texts, specific genres), manageability (feasible with limited resources), and sophistication (targeting above-surface phenomena). We hope to have shown that, for these small and deep corpora, taking the trouble of acquiring XPath and XQuery skills pays dividends.

Notes

1. This volume also contains a section in which XPath and XQuery are described in some detail.
2. For example, using a standard concordance tool, it is difficult to see how one could reliably find multiword units like *all right* highlighted in (2) when they occur within a specific context, i.e. an embedded narrative and within a direct speech (MDD) reporting segment. In contrast, either of the following XPath queries will achieve this precision:
   (A)//div[@embedLevel='EC1']//seg[@Reporting_modes='MDD']/mw/w
   (B)//mw[parent::seg/@Reporting_modes='MDD'][ancestor::div/
   @embedLevel='EC1']/w
   The first (A) is a top down approach walking down from higher-level discourse units, e.g. narrative and turn, to individual words, whereas the second (B) first locates multiword units at any point in the discourse and then restricts this list according to whether they meet the required criteria in terms of their location within the annotation structure.
3. It should be noted that these attributes, especially 'sequ' and 'depend' have been included in the annotation for convenience; they could also be calculated in XPath without any annotation. For example:
   //isoco[not(.//isoco)] – locates independent isoco elements
   //iscoco[.//isoco] – locates top level elements, equivalent to parent value for

'depend'
//isoco[parent::isoco] – locates dependent <isoco> elements, i.e., an <isoco> element embedded in another one, corresponding to the child value for the 'depend' attribute.
//isoco[count(ancestor::isoco)>1] – locates <isoco> elements embedded within at least two higher level <isoco> elements, i.e. corresponds to the grandchild value for 'depend' attribute.
Adding an id attribute to each <anaph> and <isoco> element would similarly allow XPath calculation of the position of each element within its sequence.

4. Note that the extract shown in (1) is a good reflection of the contents of the Inaugural Training Corpus (which contains the full header and just one more <s> element). The brevity of this corpus serves the purpose of enabling users to check their XQuery results against results from manual counts.
5. Here we present a few examples of this functionality in R, using the XML library:

```
> library('XML')
> doc = xmlParse('Inaugural_1_7.xml')

# to find all isoco elements that are embedded two levels deep, i.e.
depend='grandchild':
>xpathSApply(doc,'//isoco[count(ancestor
::isoco)>1]')
```

Some output:

```
[[1]]
<isoco ctype="tri" sequ="first" depend="grand-
child">
  <w id="23.34" c5="DT0">all</w>
  <w id="23.35" c5="VBB">are</w>
  <w id="23.36" c5="AJ0">equal</w>
  <c c5="PUN">,</c>
</isoco>

# to retrieve text content of first <w> in each <anaph> element
> xpathSApply(doc,'//anaph/w[1]/text()')
```

Some output:

```
[[1]]
So
```

```
[[2]]
So
[[3]]
Our
...
```

# to use R's table() function to calculate the frequency of POS categories of first words in <anaph>:

```
> table(xpathSApply(doc,'//anaph/w[1]/@c5'))
```

Some output:

```
AT0 AV0 CJC CJS DPS DT0 DTQ PNP PRP
  2   2   1   2   9   6   2  20  14
```

6. The areas in which XQuery is applied nowadays include web design, data storage and many other environments that depend on information retrieval. XQuery has grown in popularity in correlation with the growing number of XML-based markup languages, for instance the RSS format and the markup language XHTML. Its popularity has produced a large amount of software that uses the language to query XML data. Database servers therefore unsurprisingly make up a large portion of software using the language. Applications specifically designed for corpus linguists as end-users are still few and far between. See for instance, Rehm *et al.* 2008 for an application in progress. The only available application we are aware of allowing linguists to query texts is the 'eHumanities Desktop', "an online system for corpus management and analysis in support of computing in the humanities" (Gleim *et al.* 2009: 1); see http://www.hudesktop.hucompute.org/).
7. Projects that successfully use the XQuery and XPath implementation in eXist include the works cited in Section 1 but also Rehm et al.'s (2012) project that uses eXist in the analysis of heterogeneous corpora.
8. http://exist-db.org/exist/apps/homepage/index.html#download.
9. Clicking this button will display the query results not inside eXide but in a separate browser window requiring valid XML or HTML. This means that queries must be formulated inside markup in order to produce the results. For example, this query (to get all words contained in <w> elements in the IRC) is included inside an <html> and <body> tag; in the return clause the queried output (a list of the words in the speech) is contained in a <ul> (unordered list) tag. Note that all components of the XQuery code, including variables, must appear in curly brackets:

```
xquery version "3.0";
<html>
        <body>
                <ul>
                {
                for $words in collection("/db/Obama")//w
                return
                <li>{$words/string()}</li>
                }
                </ul>
        </body>
</html>
```

10. http://basex.org/products/download/all-downloads/
11. A downside to BaseX is that numerical and textual results are automatically displayed horizontally rather than vertically (which makes it hard in many applications, such as R, to process the results further; elements, by contrast, as shown in the screenshot in Figure 1, are given in vertical order). The results display can be switched from horizontal to vertical by defining the Unicode newline character as a variable in a let clause (e.g., let $nl := "&#10) and concatenating it with the results variable, as in this example (a query for words in the speech):

    ```
    for $words in //w
    let $nl := "&#10;"
    return concat($words,$nl)
    ```
12. The Editor window can be used to write not only queries but also markup, modules that visualize the tree-structure, a scatter-plot and a table that displays the results in a KWIC-like fashion.
13. Alternatively, this can be achieved by using a where clause:

    ```
    for $words in //w
    where not($words/@c5="POS")
    return
    $words/string()
    ```

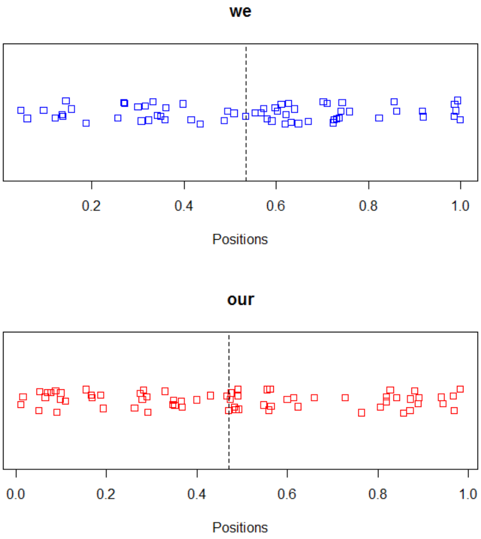
14. To compute a frequency list the full FLWOR syntax comes into play:

    ```
    let $tokens :=
    for $w in //w[not(@c5='POS')]
        return
        lower-case($w/string())
    let $types := distinct-values($tokens)
    for $type_list in $types
    let $freq := count($tokens[.=$type_list])
    where $freq >= 5
    order by $freq descending
    return
    concat($type_list,';',$freq)
    ```

    For further advanced queries targeting lexical patternings, see the XQuery Example Resource on the companion website.
15. Both *we* and *our* can be used inclusively, that is, as in the speech, to refer to speaker and addressee, or exclusively, that is, to refer to the speaker (and some company, present or absent) but not the addressee (see Levinson 1983: 69).
16. However, considering the text type at hand – inaugural address – where a newly elected leader attempts to cast him/herself as the *primus inter pares* of the community he/she was elected to lead, the very high frequency of the inclusive pronouns is much less surprising. For example, in George W. Bush's 2005 inaugural speech, the pronouns are similarly prominent: *our* and *we*, occurring 62 and 37 times respectively, account for 2.7 and 1.6 percent of all the words in that speech.
17. Initial evidence to support this hypothesis comes from a comparison of the semantic categories the nouns following *our* fall into in Obama's 2009 and Bush's 2005 inaugural addresses: in Obama's speech the nouns attributable to the category 'domestic affairs' are diverse and numerous, including, for instance, *health care*, *schools*, *colleges*, *commerce*, *gross domestic product*, and *factories*, whereas Bush's speech contains a single such noun, namely *schools* (Bush's *our* + N phrases predominantly instantiate categories such as 'history' and 'nation').
18. An intriguing question that can perfectly well be addressed by XQuery is the question of positioning: *where* in the speech are *we* and *our* used? Do they clump together in some part of the speech? Or are they distributed fairly evenly across the speech? This query returns the positions of all possessive determiners (tagged DPS):

```
for $whole_doc in //wtext
let $words_total := count($whole_doc//w[not(@c5="POS")])
for $dps in //w[@c5='DPS']
let $words_prec_dps := count($dps/preceding::w[ancestor::wtext])
let $pos := $words_prec_dps div $words_total
return
concat($pos,';',$dps/string())
```

Positions in text are calculated by the formula: (number of words before target item) divided by (total number of words in the text); as shown above, the total number of words is 2386. Thus the position of *my*, which is the first word in the speech (in *My fellow citizens*, …) is 0/2386=0, the position of the first *our*, which is the 27th word in the speech, is 26/2386= 0.0108969. The distributions of *we* and *our* are shown in the two stripcharts in the figure below:



Both *we* and *our* are fairly evenly dispersed across the whole speech. This even distribution underscores the importance of Obama's appeal to what Americans share: it is not a concern among others, but the central concern interweaving into all aspects addressed in the speech.

19. Readers may wonder why elements were not wrapped around the two rhetorical figures *as wholes*. The reason is that both anaphora and isocola often cross element boundaries. For example, anaphora occur at sentence beginnings not at sentence ends; using a single element for all repeated items would cause closing tags of <s> elements to be included within the anaphora element – a clear case of tag overlap, which is illegal in TEI-conformant XML.
20. In some cases, as in the Cesar example, the increase is not in length but in importance (cf. Leith 2011).
21. An alternative query to get at the same results in a series of steps is this:

```
for $isocola in //isoco[@ctype='tri'][@sequ='first']
let $words_colon := count($isocola//w[not(@c5="POS")])
return
$words_colon
```

    This query addresses only the first colon in the sequence; to get at the lengths of the second and third colon respectively, the restriction in the for line simply needs to be adjusted accordingly.
22. One other such type of data XQuery can process is JSON (JavaScript Object Notation), an alternative to XML (see http://www.w3schools.com/json/)
23. But since it is a complete programming language it goes without saying that XQuery can be used for statistical evaluation tasks too; however, while users of R can simply call the appropriate function, users of XQuery need to program it first.

References

Biria, Reza and Azadeh Mohammadi. 2012. The socio pragmatic functions of inaugural speech: A critical discourse analysis approach. *Journal of Pragmatics* 44: 1290–1302.

Clark, James and Steve DeRose. 1999. XML Path Language XPath Version 1.0, available at www.w3.org/TR/xpath (last accessed December 2014).

Gleim, Rüdiger, Ulli Waltinger, Alexander Mehler, and Peter Menke. 2009. eHumanities Desktop – An extensible online system for corpus management and analysis. *Proceedings of the Corpus Linguistics 2009 Conference*. In M. Mahlberg, V. González-Díaz and C. Smith (eds.). *Proceedings of the*

Corpus Linguistics Conference, available at http://ucrel.lancs.ac.uk/publications/cl2009/124_FullPaper.doc (last accessed December 2014).

Gries, Stefan Th. 2009. *Quantitative corpus linguistics with R. A practical introduction*. New York and London: Routledge.

Gries, Stefan Th. 2010. Methodological skills in corpus linguistics: A polemic and some pointers towards quantitative methods. In T. Harris and M. Moreno Jaén (eds.). *Corpus linguistics in language teaching*, 121–146. Frankfurt am Main: Peter Lang.

Gries, Stefan Th. 2013. *Statistics for linguistics with R. A practical introduction*. 2nd rev. and ext. ed. Berlin and New York: De Gruyter Mouton.

Hardie, Andrew. 2014. Modest XML for corpora: Not a standard, but a suggestion, *ICAME Journal* 38: 73–103.

Hoffmann, Sebastian, Stefan Evert, Nicholas Smith, David Lee and Ylva Berglund Prytz. 2008. *Corpus linguistics with BNCweb – A practical guide.* Frankfurt/Main: Peter Lang.

Leech, Geoffrey. 2007. New resources, or just better ones? The holy grail of representativeness. In M.Hundt, N. Nesselhauf and C. Biewer (eds.). *Corpus linguistics and the web*, 133–150. Amsterdam/New York, NY: Rodopi.

Leith, Sam. 2011. *You talkin' to me? Rhetoric from Aristotle to Obama*. London: Profile Books.

Levinson, Stephen C. 1983. *Pragmatics.* Cambridge: Cambridge University Press.

Longacre, Robert E. 1983. *The grammar of discourse*. New York: Plenum Press.

Mahlow, Cerstin, Christian Grün, Alexander Holupirek and Marc H. Scholl. 2012. A framework for retrieval and annotation in digital humanities using xquery full text and update in BaseX. *Proceedings of the 2012 ACM Symposium on Document Engineering; September 4–7, 2012, Paris, France*, 195–204. New York, NY: ACM. Available at http://kops.uni-konstanz.de/bitstream/handle/123456789/21363/mahlow_213637.pdf?sequence=2&isAllowed=y (last accessed December 2014).

O'Donnell, Matthew B., Mike Scott, Michaela Mahlberg and Michael Hoey. 2012. Exploring text-initial words, clusters and concgrams in a newspaper corpus. Special issue of *Corpus Linguistics and Linguistic Theory* 8(1): 73–101.

O'Donnell, Matthew B. and Ute Römer. 2012. From student hard drive to web corpus (Part 2): The annotation and online distribution of the Michigan Corpus of Upper-level Student Papers (MICUSP). *Corpora* 7(1): 1–18.

R Development Core Team. 2010. *R: A language and environment for statistical computing*.

R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/

Rehm, Georg, Richard Eckart, Christian Chiarcos and Johannes Dellert. 2008. Ontology-based XQuery'ing of XML-encoded language resources on multiple annotation layers. In N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis and D. Tapias (eds.). *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*. Paris: ELRA, available at http://www.lrec-conf.org/proceedings/lrec2008/pdf/139_paper.pdf (last accessed December 2014).

Rühlemann, Christoph. 2013. *Narrative in English conversation. A corpus analysis.* Cambridge: Cambridge University Press.

Rühlemann, Christoph and Matthew B. O'Donnell. 2012. Towards a corpus of conversational narrative. Construction and annotation of the Narrative Corpus. *Corpus Linguistics and Linguistic Theory* 8(2): 313–350.

Rühlemann, Christoph, Matthew B. O'Donnell and Andrej Bagoutdinov. 2013. Windows on the mind: Pauses in conversational narrative. In G. Gilquin and S. De Cock (eds.). *Errors and disfluencies in spoken corpora* (Benjamins Current Topics 52), 59–91. Amsterdam/Philadelphia: John Benjamins.

Rühlemann (eds.). *Corpus pragmatics. A handbook.* Cambridge: Cambridge University Press.

Rühlemann, Christoph and Matthew B. O'Donnell. 2015. Deixis. In K. Aijmer and C.

Scott, Mike 2010. *WordSmith tools version 5.0.* Lexical Analysis Software, Liverpool.

Scott, Mike and Christopher Tribble. 2006. *Textual patterns. Key words and corpus analysis in language education*. Amsterdam/New York: John Benjamins.

Siegel, Erik and Adam Retter. 2014. *eXist: A NoSQL Document Database and Application Platform*. Sebastopol/CA: O'Reilly.

Walmsley, Priscilla. 2007. *XQuery.* Sebastopol/CA: O'Reilly.

Watt, Andrew. 2002. *XPath essentials.* New York: Wiley and Sons.