Journal of the Text Encoding Initiative

Issue 5 | June 2013
TEI Infrastructures

PhiloLogic4: An Abstract TEI Query System

Timothy Allen, Clovis Gladstone and Richard Whaling

Electronic version
URL: http://journals.openedition.org/jtei/817
DOI: 10.4000/jtei.817
ISSN: 2162-5603

Publisher
TEI Consortium

Electronic reference
Timothy Allen, Clovis Gladstone and Richard Whaling, « PhiloLogic4: An Abstract TEI Query System », *Journal of the Text Encoding Initiative* [Online], Issue 5 | June 2013, Online since 21 June 2013, connection on 01 May 2019. URL : http://journals.openedition.org/jtei/817 ; DOI : 10.4000/jtei.817

This text was automatically generated on 1 May 2019.

TEI Consortium 2013 (Creative Commons Attribution-NoDerivs 3.0 Unported License)

PhiloLogic4: An Abstract TEI Query System

Timothy Allen, Clovis Gladstone and Richard Whaling

1. Introduction

1 A common problem in TEI-based software development is that the semantic intricacies of the XML encoding demand that the engineering team develop a custom software stack, often from scratch. While this has been a successful approach for a number of projects—to name just one example, the Perseus Digital Library has had a long run of successes with it—the duplication of effort and high cost of entry is a detriment to the TEI community as a whole. Further, the entire approach can break down when a heterogeneous corpus is aggregated from several smaller source projects. The Text Creation Partnership has not, in fact, succeed in creating rigorously interoperable texts (Unsworth and Mueller 2009; Pytlik Zillig 2009), and although the Monk Project made great strides in developing tools to normalize TEI (Unsworth and Mueller 2009; Pytlik Zillig 2009), we would observe that in the general case, the diversity of possible and extant TEI encodings demands more flexible tools. In that spirit, we will describe the design of version 4 of the PhiloLogic corpus query engine, which has a number of architectural features that address these and other obstacles.

2 Prior versions of PhiloLogic, developed in the early 2000s (Olsen 2002) and released under the Affero GPL open-source license, attempted to use extremely detailed procedural programming techniques to analyze and "guess" the best way to index a file, without a priori knowledge, by mapping a variety of common TEI XML and SGML features to an abstract data model. This worked in many common cases, but in practice, the size and complexity of the program prohibited substantial modifications or customizations. In this paper, we will describe how we have standardized PhiloLogic's abstract data model, brought it into compliance with the TEI P5 Guidelines, and leveraged contemporary XPath tools for ease of use and configurability, all while maintaining PhiloLogic's

performance and scalability characteristics. We will also demonstrate the end-user benefits of querying a data model designed to match the TEI specification, and the advantages in precision, expressivity, and ease of use over standard search engines such as Apache Solr. Finally, we will show that the generality of this architecture has substantial benefits for software reuse, and allows for powerful TEI applications to be adapted to new corpora with a minimum of custom programming. By providing an end-to-end TEI processing library, we allow developers to use the same abstract data model at all stages in the application and permit formatting software to work on a wide variety of TEI constructs and schemata. This enables developers to leverage their knowledge of their own corpora and the TEI Guidelines rather than delving into the minutiae of web frameworks and relational databases. This pragmatic benefit will lead us to discuss the more general and theoretical implications of abstraction as a TEI processing technique.

2. Data Model

3 In order to index TEI documents encoded in different ways, PhiloLogic maps such documents to an abstract data model, which is actually based on the set of "basic model classes" listed in section 1.3.2 of the TEI Guidelines (TEI Consortium 2013). These "basic model classes" can subdivide the set of all elements found in the Guidelines. This forms the basis of PhiloLogic's type system, along with a few additions.

1. Document: jointly describes the contents of the TEI Header and the `<text>` element itself.
2. Div-like: includes the component and division model classes, or other large container units.
3. Paragraph/Chunk: typically includes `<p>`, `<sp>`, `<lg>` or similar elements.
4. Sentence/Phrase: by default, sentence objects are generated by the tokenizer.
5. Words: first-class objects.
6. Parallel objects: spans of text between milestone elements forming a "parallel hierarchy" of pages, lines, or other components.

4 Of these object classes, Div-like and Sentence/Phrase are potentially recursive, to arbitrary depth (as in the TEI Guidelines), while Document, Chunk, Word, and Parallel are non-recursive. The major point of difference from the XML/XQuery data model, which treats only elements as first-class objects (W3C [World Wide Web Consortium] 2010), is that PhiloLogic treats words and sentences as first-class objects, defined either by explicit TEI markup or by user-defined tokenization patterns. This allows it to support the storage of arbitrary attributes on any object in the hierarchy, such as for common bibliographic metadata, complex morphosyntactic or analytic metadata at the word level, or most anything in between. We believe that this model is a useful compromise between the expressivity permitted by the TEI P5 Guidelines and the far stricter schema used by relational databases, and have found that it is applicable, without modification, to a large majority of extant corpora.

5 In the current state of the practice, more and more projects are moving their metadata out of TEI markup and into such formats as standoff XML files (ISO [International Organization for Standardization] 2012), SQL tables (Dik and Whaling 2008), and RESTful web services (Smith 2009). As a result, merely modeling the TEI itself can be insufficient to capture the rich ranges of data that modern projects are accumulating. To support this encoding style, PhiloLogic4 does not restrict attribute names, classes, or values to the categories defined in the Guidelines, giving developers the ability to customize and

extend the data model at will, albeit at the expense of interoperability. Ideally, all TEI constructs that contain useful metadata could be mapped to unambiguous RDF predicates, much as basic bibliographic data can be mapped to Dublin Core, and indeed we support properly namespaced object attributes for just such a use. However, in the absence of such a semantic model for the TEI Guidelines, such a practice is probably counterproductive. As we will discuss in the following section, progress in standardizing external annotation and metadata formats should greatly ameliorate this problem well before a comprehensive semantic map of the Guidelines could be completed.

3. Parser Design

6 For PhiloLogic's XML-parsing components, we had a number of interrelated design goals. We wanted to support the widest possible range of valid TEI XML documents, and we wanted to use a stream-based parsing algorithm in order to support extremely large files. Most importantly, we wanted to use a declarative syntax for specifying all behaviors of the parser.

7 PhiloLogic3 used a 3,000-line Perl SGML parser, with over a hundred regular expressions to cover the most common TEI Lite constructs; while it was extremely efficient, in practice it proved difficult to modify for other TEI variants, and its behavior was often opaque. In the last ten years, the related technologies of XPath, XQuery, and XSLT have become the tools of choice for XML processing and transformation. Although these tools are outstanding in terms of clarity and precision, they have substantial and fundamental performance disadvantages, particularly with large, heterogeneous corpora, in which the complexity of a query expression grows in proportion to the size of the corpus, and the requisite processing time thus grows exponentially, as described by Gottlob, Koch, and Pichler (2005).

8 To avoid this, we constructed a streaming, linear-time XPath evaluator capable of handling a useful subset of the full XPath specification. We control the behavior of this parser by specifying two different sets of XPath expressions: object XPaths (example 1), which map XPath expressions onto the six object types described in the previous section, and metadata XPaths (example 2), which further describe how to capture metadata attributes for each kind of object. The behavior of the parsing algorithm is simple. It walks through the tree in document order, and for each element, it first checks all the object XPaths to determine if it needs to emit a new object into the index; then it checks the current element for all metadata paths relative to all ancestor objects, and extracts content or attribute values as necessary to store in the index as a named attribute. In the case of multiple metadata matches, the attribute is only assigned to the innermost matching object.

```
XPaths = {
         "." => "doc",
  ".//front" => "div",
   ".//back" => "div",
    ".//div" => "div",
   ".//div1" => "div",
   ".//div2" => "div",
   ".//div3" => "div",
      ".//p" => "para",
     ".//sp" => "para",
  ".//stage" => "para"
      ".//s" => "sent"
      ".//w" => "word"
     ".//pb" => "page" }
```

Example 1. A set of object XPaths, expressed as a mapping of XPath expressions onto object types

```
Metadata_Xpaths = {
("doc","./teiHeader/fileDesc/titleStmt/author") => "author",
 ("doc","./teiHeader/fileDesc/titleStmt/title") => "title",
         ("doc","./teiHeader/sourceDesc//date") => "date",
                            ("doc",".@xml:id") => "id",
                              ("div","./head") => "head",
                           ("div","./head//*") => "head",
                                 ("div",".@n") => "n",
                            ("div",".@xml:id") => "id",
                          ("para","./speaker") => "who",
                             ("para","./head") => "head",
                                ("page",".@n") => "n",
                           ("sent",".@xml:id") => "id",
                           ("word",".@xml:id") => "id",
                              ("word",".@pos") => "pos",
                            ("word",".@lemma") => "lemma" }
```

Example 2. A set of metadata Xpaths, expressed as an (object-type,XPath) => field-name mapping

9 Figure 1 shows the results of our performance tests, on a disparate corpus of 1,841 TEI XML files ranging in size from a few kilobytes to five megabytes. The average parse speed over the whole collection is 16.422 microseconds per byte. In the bottom quartile, containing the 461 smallest files, the average parse speed was 17.399 microseconds per byte, whereas in the top quartile, the average parse speed was 15.261 microseconds; we believe the slight disparity is due to memory-management and garbage-collection runtime behaviors, rather than a better-than-linear performance curve. Although these measurements confirm our claim of linear performance characteristics, the overall speed

—60 seconds to parse a five-megabyte file—is by no means fast. This is to be expected from an ad-hoc implementation in a scripting language. However, for our purposes, aiming at universal applicability to any TEI XML file, a predictably slow implementation of a linear-time algorithm was preferable to a fast implementation of a potentially exponential one.

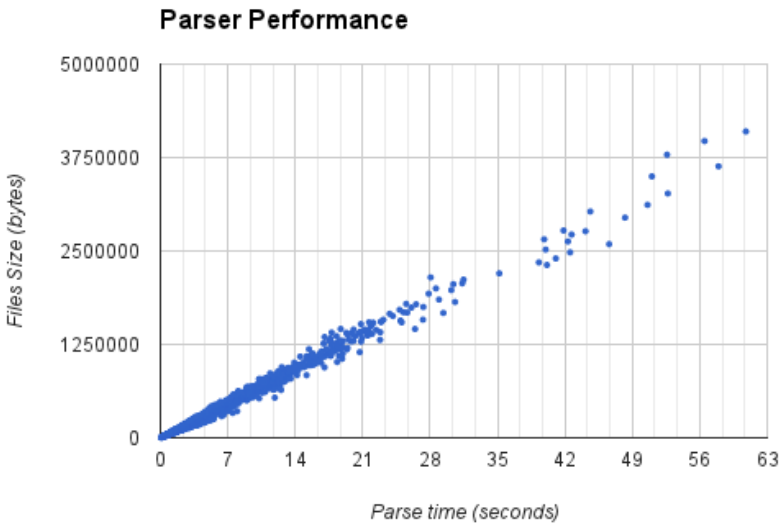

Figure 1. Performance Graph

10 A final design goal for the parser was a minimal, easily extensible, and maintainable design, in contrast to the monolithic Perl script in PhiloLogic3. At the time of writing, the parser module of PhiloLogic4 occupies just 241 lines of thoroughly-commented Python, a compactness achieved by separating out the XML/SGML lexing and internal object models into separate classes, as well as by instituting a "pipeline"-style functional API for all other operations. Thus not only the basic merging, sorting, and filtering but also more complex corpus-specific operations are encapsulated in a list of manipulations of the parser's output, and these operations maintain a common file format for all intermediate stages prior to final index compression.

11 One of the great strengths of this architecture is that it facilitates integration of external resources. For example, while PhiloLogic can extract syntactic tokens if already encoded in the text (using the metadata XPaths as described above), most documents lack this tagging and therefore need to be tokenized using a lexical database that resolves certain problematic phenomena, such as compound words in German or contractions in pre-modern English, which may be stored in a stand-off XML file or stored as a URI-accessible resource. In this case, the post-parser filtering stage is the ideal place to perform these lookups. By separating these corpus- and language-specific procedures from the main parsing code, we can allow the widest possible set of languages and texts to be handled at the highest level of accuracy, without requiring a rewrite of the parser itself. Although we currently don't include any such modules in the standard PhiloLogic distribution, we are greatly encouraged by the emergence of complementary standards such as the W3C Open Annotation Data Model (W3C 2013), ISO MAF (ISO 2012), and LMF (ISO 2008), among others, and look forward to providing comprehensive support for annotation features in a future release.

4. Query Engine

12 The exact mechanisms for querying the kind of hierarchical index structure described above are well beyond the scope of this article, although the technical problems are for the most part solved (see Witten, Moffat, and Bell 1999 and Grün 2010 for details). However, the user interface for specification of such a query remains a research question. A system like ANNIS (Rosenfeld 2010; Zeldes et al. 2009), designed for treebank queries, provides users with an extremely powerful query language for specifying intricate tree structures, assisted by a graphical query-editing tool; however, it can be very difficult to learn. Likewise, form-oriented user interfaces for complex TEI search engines tend to require the user to have substantial a priori knowledge of either the markup structure of the corpus, or of the internal data model of the search engine. In particular, we found that most users didn't use PhiloLogic3's three-level object model (ARTFL Project 2010) but instead only performed full-text or bibliographic queries due to the complexity of the search form.

13 We addressed this problem in PhiloLogic4's query engine by leveraging the same declarative markup used in the parsing procedure. The query engine can use the metadata-XPath map to infer a series of nested queries from the user input, which takes the form of a "flattened" set of key–value pairs, as depicted in examples 3a and 3b. For consistency with the parser behavior, an ambiguous or global attribute, such as `@xml:id`, is assigned to the innermost object type matching the query.

```
SELECT x FROM objects
WHERE author="Sophocles"
AND who='Oedipus'
AND pos='verb';
```

Example 3a. A flat query translated before transformation, in SQL-like pseudocode.

```
SELECT x FROM words
WHERE pos='verb'
AND x is_contained_by_one_of(
  SELECT y FROM paragraphs
  WHERE who='Oedipus'
  AND y is_contained_by_one_of(
    SELECT z FROM documents
    WHERE author="Sophocles"
  )
);
```

Example 3b. The same query, transformed by inference from examples 1 and 2.

14 This consistent object model has benefits for user interface design as well, in that it greatly facilitates the construction of a faceted browsing environment. Usually, faceted browsing is applied to commercial products or bibliographic searches, where a single set of objects is gradually reduced by the application of various filters or constraints. Although this approach has substantial usability benefits over the traditional "advanced search form" design (Hearst et al. 2002), it has proven difficult to apply to the hierarchical data structures typically used in corpus query engines: adding and removing constraints from a complex hierarchical query expression is a non-trivial operation and difficult to perform consistently based on user input. PhiloLogic4 addresses this difficulty by presenting a single set of parameters for user manipulation. The one-to-one mapping of user interactions to query parameters allowed us to construct a faceted interface in only a few hundred lines of code. In the accompanying figures 2, 3, and 4, we can see how a user can select, group, and combine metadata filters at a variety of levels—header metadata, speakers names in an <sp> tag, and word-level metadata in a <w> tag—with a single, intuitive interface.

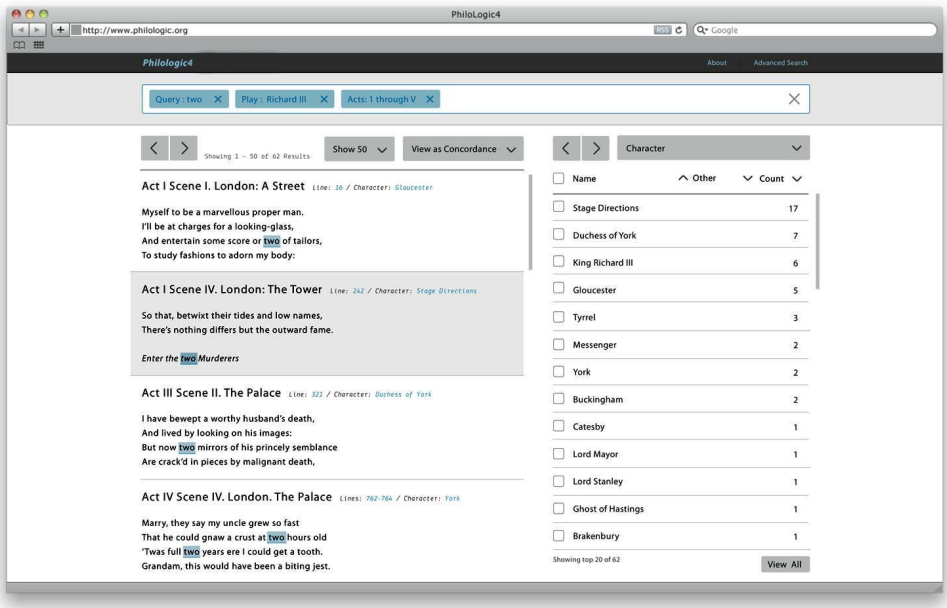

Figure 2. Faceted browsing interface with search results grouped by speaking character.

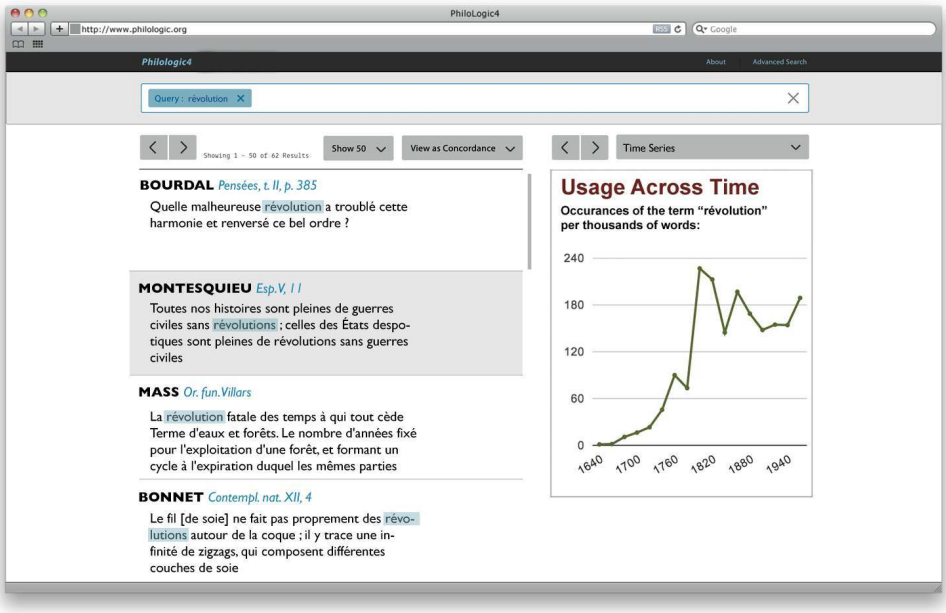

Figure 3. Faceted browsing interface showing time-series display.

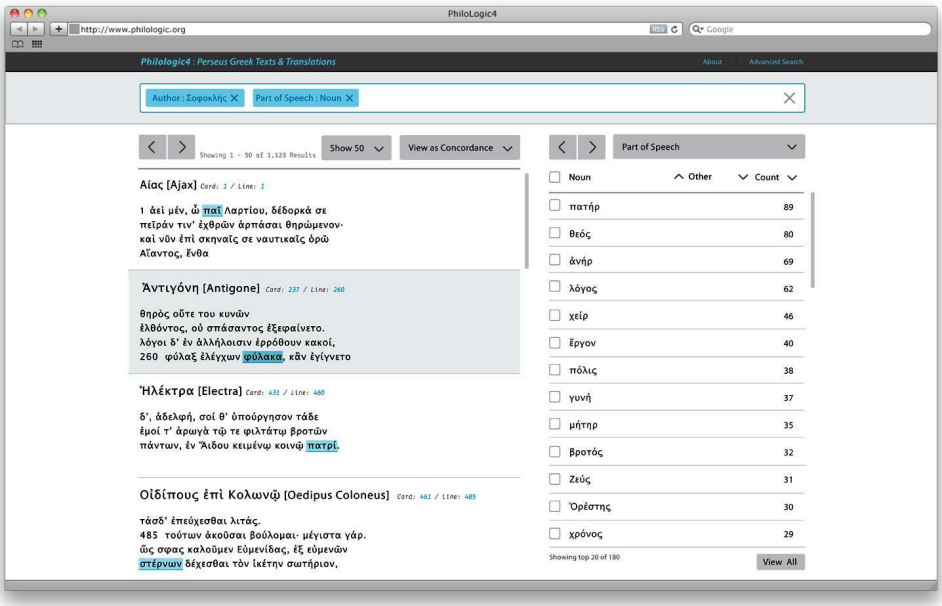

Figure 4. Faceted browsing interface showing morpho-syntactic analysis.

5. Client API

15 An unexpected outcome of our initial experiment with this "flat" query model was that it complicated the development of client software. Since any query could, in theory, return objects at any level of the document, one could easily write code that, for example, inadvertently looked for bibliographic metadata in other object types, and thus caused errors or undefined results. We had to develop a solution to handle such disparities at the programming level. Our initial attempts to develop result-formatting software used heavy introspection to infer the object type at run-time and render the results

accordingly. Eventually, we devised a more elegant approach, in which a query returned not only a list of objects but also a "wrapper" object containing the union of each object with all of its ancestors.

16 Thus, with this latter approach, if we queried `r.title`, a document-level attribute `title` on a full-text query result `r`, we wouldn't cause a type error—instead, our wrapper would implicitly scan up the hierarchy for the innermost ancestor in which the attribute is defined. This behavior demands some additional mechanism for specifying objects more granularly: in our notation, `r.div2.n` specifies the attribute `n` of the second-outermost div, and `r.word2.lemma`, the attribute `lemma` of the second word in a phrase search.[1]

17 We have found that, in practice, this kind of programming interface greatly improves the ease of developing TEI processing software. By writing formatting code against an abstract metadata model, rather than the original XML, client software can be reused with minor modifications; any changes that must be made are limited to precisely the domain of object types and attributes that the database specialist built at the parsing stage, and thus do not require specialized knowledge of XML transformation libraries or the small variations in encoding that are endemic in large corpora. For example, much of the formatting code that we used for ARTFL-Frantext, a 3,500-volume collection of texts from the 12th to 20th centuries, could be re-used for Diderot and d'Alembert's *Encyclopédie* with only the smallest changes by moving the "author" and "title" metadata fields from the document object type to the `<div1>`s and `<div2>`s. An additional benefit is that this flexibility extends to non-XML data sources as well, for the integration of data from spreadsheets, MARC records, relational databases, or other data sources as desired.

6. Conclusion

18 The design of a modern corpus query engine goes beyond the architectural considerations described in this paper: scaling and distribution behaviors, especially, are critical in the current environment of mass digital corpora. However, a conceptually sound data model is a necessary prerequisite for this sort of engineering. Furthermore, while the success of any open-source software project depends as much on the community that develops around it as on any intrinsic factors, we feel that the innovations we have described can help to expose the expressive power of TEI markup to a much broader and more general audience than has previously had effective access to it. We invite feedback from the broadest spectrum of the TEI community as we continue to develop our software, not just for the improvement of any one piece of software but also to better explore and explain the operational and semantic models that underlie the markup we work with daily.

BIBLIOGRAPHY

ARTFL Project. 2010. "Developing Forms - PhiloLogic3." Accessed March 28, 2013. https://sites.google.com/site/philologic3/wiki/development-forms.

Dik, Helma, and Richard Whaling. 2008. "Bootstrapping Classical Greek Morphology." Paper presented at Digital Humanities, Oulu, Finland, June 25–29, 2008.

Gottlob, Georg, Christopher Koch, and Reinhard Pichler. 2005. "Efficient Algorithms for Processing XPath Queries." *ACM Transactions on Database Systems* 30(2): 444–91. doi:10.1145/1071610.1071614.

Grün, Christian. 2010. "Storing and Querying Large XML Instances." PhD diss., University of Konstanz.

Hearst, Marti, Ame Elliott, Jennifer English, Rashmi Sinha, Kirsten Swearingen, and Ka-Ping Yee. 2002. "Finding the Flow in Web Site Search." *Communications of the ACM* 45(9): 42–49. doi:10.1145/567498.567525.

ISO (International Organization for Standardization). 2012. "Language Resource Management — Morpho-syntactic Annotation Framework (MAF)." ISO 24611:2012. Geneva: ISO.

ISO (International Organization for Standardization). 2008. "Language Resource Management — Lexical Markup Framework (LMF)." ISO 24613:2008. Geneva: ISO.

Olsen, Mark. 2002. "Rich Textual Metadata: Implementation and Theory." Paper presented at the Joint International Conference of the Association for Literary and Linguistic Computing and the Association for Computers and the Humanities, Tübingen, Germany, July 24–28, 2002.

Pytlik Zillig, Brian L. 2009. "TEI Analytics: Converting Documents into a TEI Format for Cross-collection Text Analysis." *Literary and Linguistic Computing* 24, 187–92.

Rosenfeld, Viktor. 2010. "An Implementation of the Annis 2 Query Language." Technical report, Humboldt-Universität zu Berlin.

Smith, Neel. 2009. "Citation in Classical Studies." *Digital Humanities Quarterly* 3(1). http://www.digitalhumanities.org/dhq/vol/3/1/000028/000028.html.

TEI Consortium. 2013. "TEI P5: Guidelines for Electronic Text Encoding and Interchange." Edited by Lou Burnard and Syd Bauman. Version 2.3.0. Last updated January 17, 2013. http://www.tei-c.org/release/doc/tei-p5-doc/en/html/index.html.

Unsworth, John, and Martin Mueller. 2009. "Monk Project Final Report." Technical report. Accessed March 28, 2013. http://www.monkproject.org/MONKProjectFinalReport.pdf.

Witten, Ian H., Alistair Moffat, and Timothy C. Bell. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images*. 2nd ed. San Francisco: Morgan Kaufmann.

W3C (World Wide Web Consortium). 2013. "Open Annotation Data Model." Community Draft, 08 February 2013. Edited by Robert Sanderson, Paolo Ciccarese, and Herbert Van de Sompel. http://www.openannotation.org/spec/core/.

W3C (World Wide Web Consortium). 2010. "XQuery 1.0 and XPath 2.0 Data Model (XDM)." 2nd ed. W3C Recommendation 14 December 2010. Edited by Anders Berglund, Mary Fernández, Ashok

Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh. http://www.w3.org/TR/xpath-datamodel/.

Zeldes, Amir, Julia Ritz, Anke Lüdeling, and Christian Chiarcos. 2009. "Annis: A Search Tool for Multi-layer Annotated Corpora." Proceedings of the Corpus Linguistics Conference, University of Liverpool, UK, July 20–23, 2009, article no. 358. http://ucrel.lancs.ac.uk/publications/cl2009/.

NOTES

1. Since div objects are potentially recursive, whereas words are not but are often related to one another in a phrase, we define distance in different ways for the various object types, hence the semantic difference between `div2` and `word2`. We may revise and expand this syntax in the future for more clarity.

ABSTRACTS

A common problem for TEI software development is that projects develop their own custom software stack to address the semantic intricacies present in a deeply-encoded TEI corpus. This article describes the design of version 4 of the PhiloLogic corpus query engine, which is designed to handle heterogeneous TEI encoding through its redesigned abstract data model. We show that such an architecture has substantial benefits for software reuse, allowing for powerful TEI applications to be adapted to new corpora with a minimum of custom programming, and we discuss the more general and theoretical implications of abstraction as a TEI processing technique.

INDEX

Keywords: corpus query, indexing, search, user interface, parsing, web frameworks

AUTHORS

TIMOTHY ALLEN

Timothy Allen is a project manager at the ARTFL Project, University of Chicago.

CLOVIS GLADSTONE

Clovis Gladstone is a PhD candidate in French literature at the University of Chicago whose dissertation work focuses on 18th-century intellectual history. Since 2008 he has been working for the ARTFL Project, where he has explored new methods of retrieving relevant data from large databases and participated in the development of the online Dictionnaire Vivant de la Langue Française (DVLF).

RICHARD WHALING

Richard Whaling is an Emerging Technologies Developer at the University of Chicago IT Services and the ARTFL Project.