Journal of the Text Encoding Initiative

Issue 8 | December 2014 - December 2015
Selected Papers from the 2013 TEI Conference

TeiCoPhiLib: A Library of Components for the Domain of Collaborative Philology

Federico Boschetti and Angelo Mario Del Grosso

Electronic version
URL: http://journals.openedition.org/jtei/1285
DOI: 10.4000/jtei.1285
ISSN: 2162-5603

Publisher
TEI Consortium

Electronic reference
Federico Boschetti and Angelo Mario Del Grosso, « TeiCoPhiLib: A Library of Components for the Domain of Collaborative Philology », *Journal of the Text Encoding Initiative* [Online], Issue 8 | December 2014 - December 2015, Online since 23 September 2015, connection on 19 April 2019. URL : http://journals.openedition.org/jtei/1285 ; DOI : 10.4000/jtei.1285

TeiCoPhiLib: A Library of Components for the Domain of Collaborative Philology

Federico Boschetti and Angelo Mario Del Grosso

1. Introduction

1 The TeiCoPhiLib library is a collection of components currently implemented in Java (JSR 270), which parses documents encoded according to a basic subset of TEI tags defined in an ODD file[1] and creates an object-oriented data structure in order to handle the textual content and its processing. The overall architecture is based on the well known Model-View-Controller (MVC) pattern, which separates the representation of data from the rendering and management of the content for the sake of flexibility and reusability. TeiCoPhiLib maps the structured document onto an aggregation of objects. The library enables the visualization through a web browser by instantiating a collection of widgets rendered on the client through standard web technologies. Specifically, the server-side environment jointly processes data and visualization templates,[2] and generates HTML pages rendered on the client. Special components are devoted to monitoring the behavior and interactions among the objects generated from the input TEI documents.

2 In distributed and collaborative environments, the maintenance of links and relations among editable XML documents is a challenging task (Di Iorio et al. 2009; Peroni, Poggi, and Vitali 2013; Schmidt and Colomb 2009). Indeed, this kind of environment must preserve the referential integrity among interrelated textual units and the consistency among interlinked contents (Schmidt 2014; Barabucci et al. 2013; Ronnau, Philipp, and Borgho 2009). It is worth noting that both texts and related annotations may change asynchronously. Accordingly, the problems with maintenance have an increasing relevance in social editing (Siemens et al. 2012) and in general in collaborative philology. This emerging field concerns the social activity of scholars focused on shared philological tasks (such as scholarly editing and collaborative annotation) through a cyberinfrastructure (Terras and Crane 2010; Crane, Seales, and Terras 2009). In order to face this challenge, our approach exploits software engineering techniques illustrated in section 3.3, which explains the TeiCoPhiLib design patterns.

3 For this reason, the design of TeiCoPhiLib widely leverages the stand-off approaches provided by the TEI Guidelines, that is, both the reference to plain text offsets and the reference to nodes denoted by the `@xml:id` unique identifiers (TEI Consortium 2015; Wittern, Ciula, and Conal 2009; Pierazzo 2015). Consequently, the design of the components aims at the separation of concerns through four distinct layers: (1) textual structure; (2) semantics; (3) style; and (4) behavior, in order to ensure modularity, scalability, and flexibility.

2. Background

4 The main benefit of XML, and especially of the TEI Guidelines (TEI Consortium 2015), resides in simplicity, flexibility, readability, and customizability, with the assurance of a formal approach for validating the marked-up data. Consequently, XML provides a standard way to define a set of tags (vocabulary) for specific purposes. Moreover, the cluster of technologies associated with XML[3] allows us to process, query and publish structured documents.

5 Several frameworks and initiatives have been developed over the years for handling XML, achieving great results and benefits for both scholars and developers. Among others, the open-source general-purpose framework Cocoon[4] and the native XML database eXist-db[5] deserve to be mentioned. Specifically for TEI-annotated documents, TUSTEP,[6] TEIBoilerplate,[7] TXM,[8] and TAPAS[9] are prominent projects.

6 For all of these initiatives, the transformation from an XML document structure to another format by XSLT can be considered the focal point.

3. Method

3.1 Flexibility and Reusability

7 A document-oriented approach can be complemented by an application/API-oriented approach for the development of textual analysis tools. We are adopting a top-down design integrated with bottom-up processes (Del Grosso and Boschetti 2013), which allows us to generalize, extend, and refactor the overall architecture as new requirements and common issues emerge from use cases under development. The design is top-down because we are defining both the general abstract framework and the mechanisms that allow us to implement new functionalities according to emerging needs. On the other hand, our library also adopts a bottom-up approach because we apply refactoring strategies to adapt existing components implemented in our previous projects to the general framework, extending the framework.

8 The library of components is designed by exploiting object-oriented methods and processes such as analysis of requirements, definition of the domain entities, separation of concerns, information hiding, and software reusability and extensibility (Fowler 1996). Extensive use of design patterns (i.e., recurring solutions to common problems within a given context [Gamma et al. 1995]) facilitates the achievement of these goals.

9 Agile software development[10] and use case–driven modeling (Rosenberg and Stephens 2007) ensure the progressive enhancement of old functionalities and the development of new ones. The main principles of agile software development that we adopt are: (1) individuals and communication are more important than processes and tools; (2) documentation and design must be accessible to everybody all the time; (3) software development starts as soon as possible; (4) changes and refactoring are part of the design and the development process; (5) all lab team members participate in all presentations; (6) software is organized in short releases and divided into short iterations; (7) results are validated by domain expert collaborations and test-driven development (both unit tests and acceptance tests). The continuous integration and release are

supported by open source Integrated Development Environments (IDEs) like Eclipse or NetBeans and by a software configuration management tool such as SVN or Git for versioning and revision control.

10 The aforementioned paradigm is applied in the TeiCoPhiLib library by (1) the implementation of a flexible importing and normalization module in the pre-processing phase, which ensures a coherent abstraction model of the resources; (2) the definition of the functional specification by designing the objects and by declaring the application interfaces; (3) the export of the information encapsulated in the objects into different data formats, in order to enable data integration and data exchange.

3.2 Separation of Concerns

11 First of all, objects that represent the whole document or interrelated documents are initialized by parsing the original TEI document(s) and by creating a new data structure, which decouples the orthogonal information conveyed by the XML elements: (1) textual structure, (2) semantics, (3) style, and (4) behavior. It is important to point out that the new data structure is the result of transformations (by XSLT DOM transformations or SAX event-driven transformations) managed during the parsing process. Thus, the current implementation of the TeiCoPhiLib exposes methods that parse the XML file and create Java objects. The resources are stored and maintained in a native XML database management system (i.e., eXist-db). The APIs and services provided by Lucene, a software library developed and hosted by the Apache Foundation, have been used for indexing the textual data.

12 For instance, the information conveyed by the following TEI snippet is distributed among the appropriate Java objects that handle the four levels described above:

```
<div type="chapter" n="1" style="font-variant:normal">
 [...]
 <p xml:lang="ita">
  <lb n="1"/>Io nacqui veneziano ai
  18 ottobre del 1775, giorno
  <lb n="2"/>dell'evangelista san Luca;
  e morrò per la grazia di Dio
  <lb n="3"/>italiano quando lo vorrà
  quella Provvidenza che governa
  <lb n="4"/>misteriosamente il mondo.</p>
 [...]<milestone type="page" n="2"/>[...]
</div>
```

13 The parsing process concerns the following aspects:

1. *Textual structure*. The same document originally structured paragraph by paragraph for literary analysis can easily restructured page by page for layout analysis and for comparison with the original page image. Semantics, style, and behavior are represented by objects separated from (but linked to) the nodes of the DOM tree.
2. *Semantics*. At the semantic level, both attributes (such as `@type`) and tag names (such as `<p>`) are processed in the same way and linked to the related DOM node.
3. *Style*. The style is managed by separated renderers, which point to textual positions affected by stylistic features. For instance, the information extracted from the `@style` attribute is used to instantiate the Java objects devoted to managing the rendering information.
4. *Behavior*. Behaviors are handled by objects that process textual resources according to the current state of the data structure and the rules to manage such a state. For example, a hyphenator performs its tasks according to the language of the textual data (encoded in the original TEI file, e.g., by `@xml:lang="it"`) and the related hyphenation rules (such as the hyphenation rules for the Italian language, managed by the hyphenator bundles).

14 The object-oriented representation of the document allows data to be processed dynamically, taking account of its physical and logical structure, in an attempt to overcome the multiple hierarchies issue. This means that the data model of the library has a decoupled and abstract structure which can be serialized in any available file format, including all standard TEI-

compliant approaches. Consequently, the TeiCoPhiLib document entity keeps structure and logical information in an independent but related aggregate of objects (figure 3). Furthermore, output modules or visitors (figure 2) can traverse and serialize the object representation of the document to a file. In particular, modules for TEI files define the marshalling process, which encompasses the operation to obtain the encoded XML files. In this way the system provides the actual representation of the document.

3.3 Design Patterns

15 The overall architecture of the library is based on several design patterns, according to the object-oriented paradigm (Gamma et al. 1995; Buschmann, Henney, and Schmidt 2007).

16 Design patterns were introduced in software engineering in order to provide a common solution for a recurring problem in a specific context. A design pattern defines the instantiation policies, the structure, or the behavior for an aggregate of objects that cooperate to provide a complex but recurrent functionality, such as the creation of polymorphic entities, the management of decoupled modules, or the selection at run time of the most suitable algorithm for the current task. The general idea of object-oriented patterns is to encapsulate functionality and data inside an efficient and flexible collection of classes. The current implementation of the prototype exploits the Java programming language technologies.

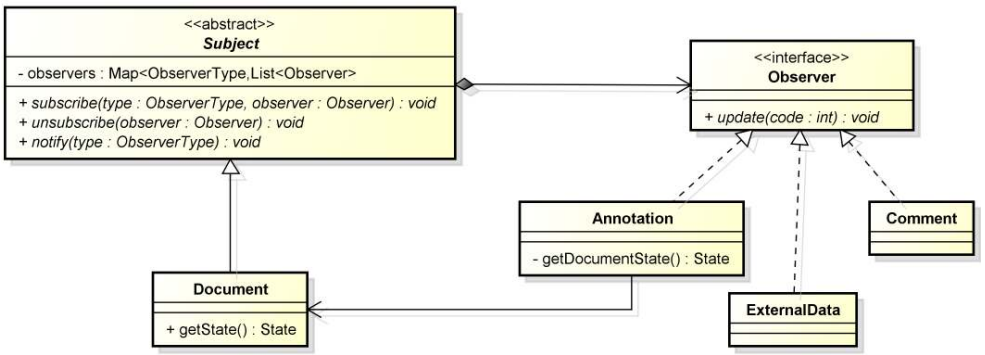
1. The *Model-View-Controller* (MVC) pattern (Burbeck 1992) determines the architecture of the library by separating the internal representation of the data from the rendering and the behavioral purposes.

2. The *Factory* pattern allows designers to enhance the object creation procedure by means of special classes (i.e., the factory classes) that guarantee abstract coupling among system modules (i.e., the object relationships do not reference implementing classes). As a matter of fact, the foregoing design makes it possible to programmatically reference abstract objects independently from the run-time instances which actually perform the task. This design pattern lets applications change the implementation of a class in a flexible way. In our case, for instance, a document object has textual content maintained in a specific DOM data structure transparent to the user. The client agent is able to manipulate and process the document independently of its internal DOM representation. The algorithms and processing keep the state of the data structure coherent by updating the DOM representation in a transparent way.
3. The *Builder* pattern is used to initialize and populate the document data structure. Together with the factory pattern, it hides the real type of the objects from the user agents and maintains the state consistency of the interconnected information. In this way, in the library initialization process, it is possible to create different data structures for different aims, in a way that is completely transparent to the user agents. For instance, a Builder oriented to the layout analysis can restructure the information parsed from the TEI input document.
4. The *Composite* pattern is the core of the data structure (figure 3). The document object is defined as an aggregation of hierarchical entities with the same data type. The hierarchy maps either the DOM structure of the original XML-TEI document or the structure of one of its transformations based on an XSLT input parameter. Thanks to this pattern, an efficient object-oriented structure, sketched through the UML class diagram on the right of figure 3, represents the whole/part relationships among the objects in the data structures.

5. The *Strategy* pattern implements different operations in different ways based on the object type or on specified parameters. For example, the building process uses different strategies, which are driven from specific features given through property files. In the previous example, the original TEI page milestone is represented by an element node in the DOM internal structure; conversely, the original TEI element for paragraph can be represented by a milestone. Furthermore, the Strategy pattern is useful for rendering the same data in multiple views in different contexts or processing the same data with different algorithms.

6. The *Observer* pattern provides a mechanism for handling dependencies among interrelated objects. This ensures that when a change occurs, the overall state is synchronized and updated. For example, if an edit operation deletes some values in the document, all related structures are notified and updated accordingly. The library organizes the entities derived from the original document information through the stand-off approach. In this way the document structure is separate from its semantics, style, and behavior.

Figure 1. Class diagram of the Observer pattern designed for the TeiCoPhiLib.



Several modules of the library need synchronized data. In particular, annotations and comments need to be notified if an event occurs that changes the state of the referenced object. This problem is well known in the context of graphical user interfaces, where embodied components communicate by message exchange according to a standard protocol. The Observer pattern, which is a behavioral technique, solves the aforementioned problem in an elegant and easy way. TeiCoPhiLib exploits the Observer pattern to manage communication among the object-oriented representation of the document and the encapsulated data that point to it. Figure 1 shows the UML class diagram of the mechanism designed for the library. The pattern involves two interfaces: (6a) Subject and (6b) Observer. These two entities provide the flexibility to implement a decoupled notification mechanism. The Subject provides a registration procedure for the Observer object, and the Observer object provides a standard method allowing the Subject to notify it. TeiCoPhiLib defines objects that can change, such as the Document data type, and objects that need to be notified, such as the Annotation or the Comment data types. Consequently, the Document

class implements the Subject interface, whereas the Annotation and Comment classes implement the Observer interface. The following simplified Java snippet illustrates this concept programmatically.

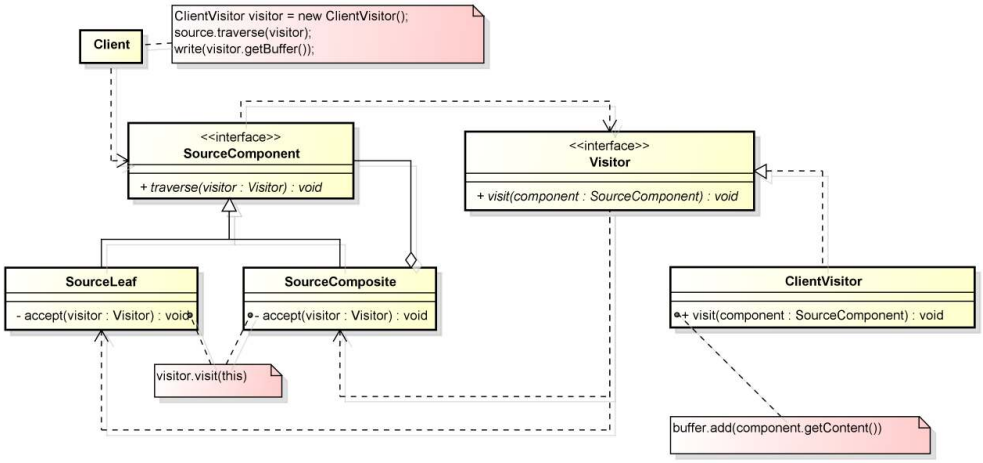
```
Observer annotation = new Annotation();
Observer comment = new Comment();

Subject teiDocument = new Document();
teiDocument.subscribe(ObserverType.ANNOTATION, annotation);
teiDocument.subscribe(ObserverType.COMMENT, comment);

// some processing that modifies the teiDocument object
// accordingly the observers have to be notified
//[...]
teiDocument.notify(ObserverType.ANNOTATION);
```

7. The pattern *Visitor* allows the user agents to gather data which are stored in different object data fields and enables the reconstruction of a consistent and homogeneous view. The stand-off mechanism implies a distributed and interrelated structure where information is maintained across various objects. For example, during the exporting phase the object-oriented representation of the document is processed in order to produce a valid TEI XML document according to a schema selected by the user.

Figure 2. Class diagram of the Visitor pattern designed for the TeiCoPhiLib.



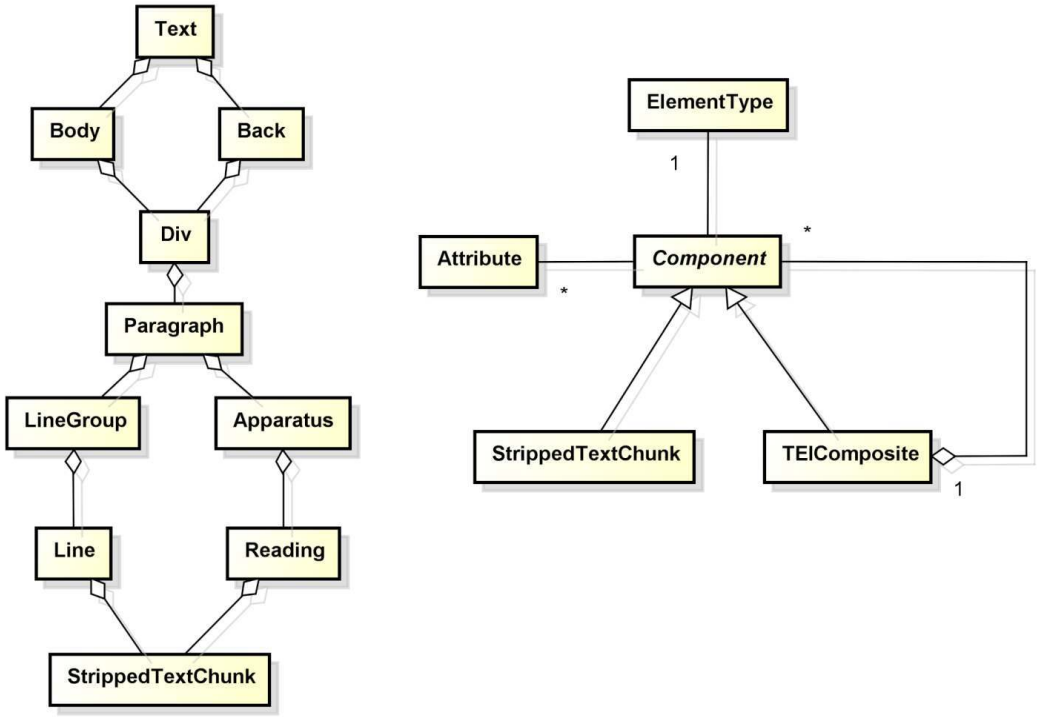
The hierarchical nature of the document representation facilitates the data structure traversal in a flexible and customizable way. The Visitor pattern provides a mechanism to extend the functionality of the TeiCoPhiLib by allowing components to perform a client-supplied operation on each node of the document hierarchy. Figure 2 shows how a client of the data model can traverse the document tree in order to write its textual content. The mechanism supplies the actual visitor instance by means of a custom operation. The current implementation of the Visitor interface is then used by the entities of the hierarchy which are unaware of the actual behavior of the visitor instance. In this way, the Visitor pattern provides a procedure to operate on the representation of the document by making available to the clients a point of extensibility (Martin 2000).

17 An example should clarify the aforementioned architecture. The client application that uses TeiCoPhiLib APIs invokes the building method of the abstract Builder class. Moreover, the resulting document object is a concretization of an abstract class representing the current structure of the TEI-encoded resource, as illustrated in the following Java statement:

```
Document teiDocument = AbstractBuilderFactory.buildDocument(new
File("features.properties"),new File("teiDocument.xml"));
```

18 The Builder object needs two input files: (1) a property file containing the suitable configuration for the instantiation of the concrete objects; (2) the TEI XML file to parse. The state of the internal representation of the document is composite-component–based. In this way each single element is handled by the TEIComposite object shown on the right-hand side of figure 1, which represents each node of the hierarchical DOM structure. Leaves are instances of the StrippedTextChunk class (figure 3). Methods in such a structure make it possible to manipulate the content and the structure of the resources.

Figure 3. Parsed TEI document (left), mapped into a Composite pattern structure (right).



19 Finally, the data structure is an object-oriented representation of the entities in the real domain of the digital document, but the storage platform/paradigm could actually be relational, hierarchical, semi-structural (XML), or network/graph–structured (Hohpe and Woolf 2004).

20 The UML diagram on the left side of figure 1 shows the structure of a TEI document DOM representation. Objects, in this case, exactly map the tags of the TEI-encoded file format. The UML diagram on the right side illustrates the composite-components design. This diagram maps the same aforementioned information and structure, but using a flexible and recursive design. Each TEIComposite object reflects a TEI element and each child is recursively a TEIComposite or a textual element.

4. Case Studies

21 The case studies illustrated below have been implemented with the components already developed for our library.

4.1 Euporia: Visualization, Editing, and Annotation of Parallel Texts for Didactic Purposes

22 Euporia is a project aimed at visualizing, editing, and annotating bilingual texts displayed in parallel. The original digital resources are stored and maintained in authoritative digital libraries available online, such as the Biblioteca Italiana and the Perseus Digital Library, or they are downloaded from social proofreading websites, such as WikiSource, and subsequently processed and marked up in TEI. Some examples of Greek and Latin texts potentially alignable or actually aligned with their Italian translations are shown in table 1.

23 As mentioned in section 3, different subcollections of texts that must be aligned may provide or omit some extratextual information (such as line number or page number) and they may organize texts in different ways (for instance, lines can be grouped or not inside `<lg>` elements). For this reason, the XSD schema (which is expected to be a subset of the general TEI schemas) is generated *a posteriori* from the actual text subcollections. This approach can be considered complementary to the TEI Roma approach, a kind of reverse engineering, which also allows us to generate the ODD file. Studying the schemas, XSLT transformations are created in order to deal only with relevant information and canonical formats processed by the appropriate Aligner. Currently only

the `SpeechAligner` for dramatic texts has been implemented: correspondences between the Italian translation and the original Greek text are automatically injected with the `@corresp` attribute (see row B in table 1) and misalignments must be manually corrected.

Table 1. Parallel texts.

A) Perseus Digital Library	Digital Library of Biblioteca Italiana

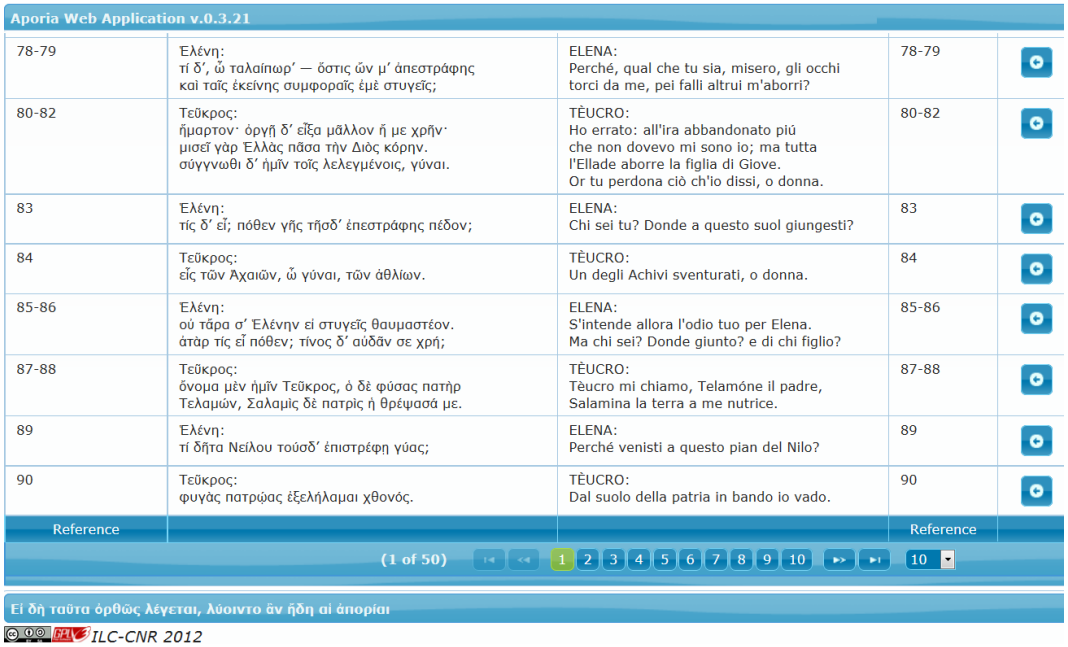
```
  <!-- www.perseus.tufts.edu -->
  <div1 type="Book" n="2" org="uniform"
sample="complete">
   <milestone ed="p" n="1" unit="card"/>
   <l>Conticuere omnes, intentique ora
tenebant.</l>
   <l>Inde toro pater Aeneas sic orsus ab
alto:</l>
   <l>Infandum, regina, iubes renovare
dolorem,</l>
   <l>Troianas ut opes et lamentabile
regnum</l>
   <l n="5">eruerint Danai; quaeque ipse
miserrima vidi,</l>
   <l>et quorum pars magna fui. Quis talia
fando</l>
   <l>Myrmidonum Dolopumve aut duri miles
Ulixi</l>
   <l>temperet a lacrimis? Et iam nox
umida caelo</l>
   <l>praecipitat, suadentque cadentia
sidera somnos.</l>
   <l n="10">Sed si tantus amor casus
cognoscere nostros</l>
   <l>et breviter Troiae supremum audire
laborem,</l>
   <l>quamquam animus meminisse horret,
luctuque refugit,</l>
   <l>incipiam.<milestone ed="P"
unit="para"/>Fracti bello fatisque
repulsi</l> [...]
  </div1>
```

```
  <!-- www.bibliotecaitaliana.it -->
  <div1 sample="complete"> [...] <lg
sample="complete">
    <l>Tutti ammutiro e s'affisaro.
Enea</l>
    <l>Da l'alto seggio
incominciava. Infando,</l>
    <l>O Regina, è il dolor cui tu
m'imponi</l>
    <l>Ch'io rinnovelli. I' dovrò
dir da' Greci</l>
    <l>I teucri averi e il miserando
regno</l>
    <l>Come fosser diserti; io dire
i casi</l>
    <l>Tristissimi dovrò, cui vidi
io stesso</l>
    <l>E di che fui gran parte. E
qual potrebbe</l>
    <l>O Mirmidone o Dolope o
seguace</l>
    <l>Del fero Ulisse rattenere il
pianto</l>
    <l>Tai cose in ragionando? E già
dal cielo</l>
    <l>Precipita la notte umida, e
gli astri</l>
    <l>Vanno in cader persuadendo il
sonno.</l>
    <l>Ma se tanto desio nel cor ti
siede</l>
    <l>De l'ascoltar de' nostri casi
e in breve</l>
    <l>Udir di Troia l'ultima
sciaura,</l>
    <l>Benchè pur del pensiero io mi
spaventi,</l>
```

	<l>Comincerò. Dopo tant'anni infine</l> [...] </lg> [...] </div1>
B) Perseus Digital Library	**Euporia Resources**
<sp id="sp_grc_0002"> <speaker>Ἑλένη</speaker> <lb/>τί δ, ὦ ταλαίπωρ' – ὅστις ὢν μ' ἀπεστράφης <lb/>καὶ ταῖς ἐκείνης συμφοραῖς ἐμὲ στυγεῖς; </sp>	<sp id="sp_ita_0002" corresp="#sp_grc_0002"> <speaker>ELENA</speaker>: <lb/>Perché, qual che tu sia, misero, gli occhi <lb/>torci da me, pei falli altrui m'aborri? </sp>

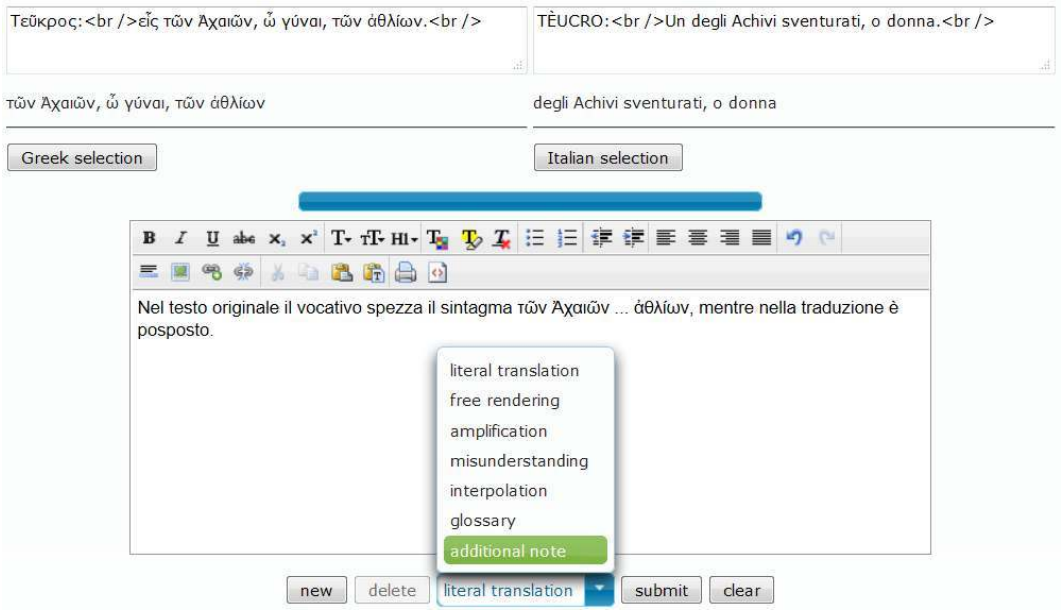
24 Parallel texts are visualized and managed through EuporiaWebApp (figure 4), which is a server-side Java web application compliant with the JSR 314 specification intended for educational purposes. Students, the end users of Euporia, are allowed to query texts, both jointly and independently, through multilingual or monolingual keywords.

Figure 4. Euporia.



25 Annotations can also be associated with linked chunks of text (such as a sentence and its translation: see figure 5) or with single, independent chunks of text (such as a single word of the original text). Annotations and their pointers to the text are stored as stand-off markup. Currently, the adopted stand-off linking mechanism uses only `@xml:id`, but the use of XPointer is under evaluation.

Figure 5. Annotation in Euporia.

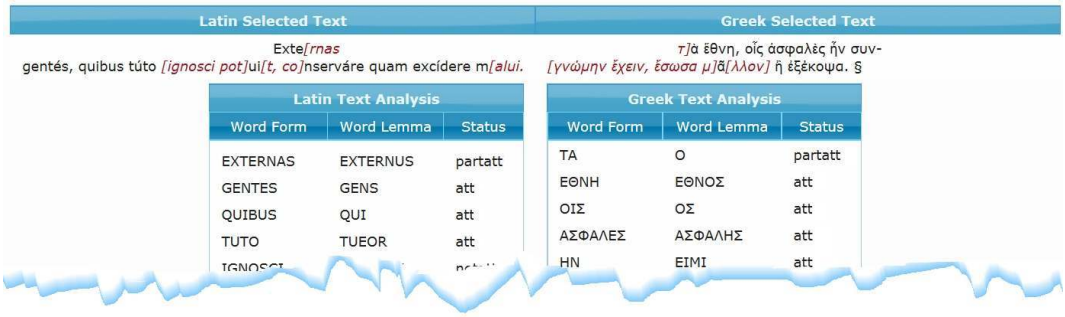


4.2 Aporia: Adapting the Parallel Text Framework to Specific Scientific Requirements

26 Aporia is the enhanced version of Euporia, intended for research purposes. Accordingly, the Parallel Text framework has been adapted and extended to meet specific scientific requirements (Bozzi 2013). An experimental case study has been performed on Theodor Mommsen's edition of the *Res Gestae Divi Augusti* (1883), in order to examine the complementarity of attested fragments of text in the Latin inscription of the *RGDA* and the related attested fragments in the Greek translation (Lamé 2012). Attested and conjectural parts from Mommsen's critical edition have been marked. As shown in figure 6, the feature "status" (attested / partially attested / conjectural) is not only visualized in different colors (a CSS stylesheet is enough for this task), but also available in query masks to filter the results of a query (for instance, "find only attested or partially attested words") and in tables of results.

27 Unlimited stand-off layers of analysis can be added (such as morphological, syntactic, and semantic analysis), at different levels of granularity (for example, at the level of words for morphological analysis and at the level of sentences for semantic analysis). The layers of annotation are added through the web application and stored in the XML database. Moreover, they can be manually encoded and visualized through the web application.

Figure 6. Aporia.



4.3 Saussure Project: Supporting Genetic Criticism

28 The Saussure Project exploits the flexibility of Aporia in order to adapt the system to the study of Saussurean autographs, making author's variants searchable and creating multilingual indexes of ancient terms studied by the linguist.

29 Instead of showing linked texts in parallel, the system shows the image of the manuscript and the related transcription (figure 7).

Figure 7. Saussure Project.

5. Conclusion

30 The TeiCoPhiLib is a work in progress focused on the creation of a library of software components aimed at managing a limited subset of TEI tags used in the domain of collaborative philology. Because of the increasing complexity of annotations and the multiple usages of the same texts in collaborative environments, stand-off annotation and dense mark-up make it challenging to keep annotated documents readable and manageable. While annotation focuses on types of texts (such as poetic, dramatic, and with or without critical apparatus), software development focuses on abstraction of data structures and behaviors related to those texts (such as searching them in parallel, filtering by morphological features, and comparing text and image).

31 Reusable software components promote the management of stand-off annotation at any level (such as editing, searching, or visualizing), improving the experience of the annotation and use of TEI documents.

32 The document parsing in the current Java implementation takes place on the server side, where the Java virtual machine runs within the web application environment.

33 The marshalling and unmarshalling process handles the serialization of the object representation of the TEI document, in order to store and retrieve data on the filesystem or in native XML databases, such as eXist-db.

34 Performance measurement tools such as JMeter will help to optimize the performance of the library components.

35 Software currently under development will be available on GitHub at https://github.com/CoPhi/cophilib.

BIBLIOGRAPHY

Barabucci, Gioele, Angelo Di Iorio, Silvio Peroni, Francesco Poggi, and Fabio Vitali. 2013. "Annotations with EARMARK in Practice: A Fairy Tale." In *DH-CASE '13: Proceedings of the 1st International Workshop on Collaborative Annotations in Shared Environment: Metadata, Vocabularies and Techniques in the Digital Humanities*, article no. 11. New York: ACM. doi:10.1145/2517978.2517990.

Bozzi, Andrea. 2013. "G2A: A Web Application to Study, Annotate and Scholarly Edit Ancient Texts and Their Aligned Translations." *Studia graeco-arabica* 3:159–71. http://www.greekintoarabic.eu/uploads/media/BOZZI_SGA_3-2013.pdf.

Burbeck, Steve. 1992. "Applications Programming in Smalltalk-80TM: How to Use Model-View-Controller (MVC)." Last modified March 4, 1997. http://www.math.sfedu.ru/smalltalk/gui/mvc.pdf.

Buschmann, Frank, Kevlin Henney, and Douglas C. Schmidt. 2007. *Pattern Oriented Software Architecture*. Vol. 5, On Patterns and Pattern Languages. Wiley Software Patterns Series. Chichester, England: John Wiley & Sons.

Crane, Gregory, Bridget Almas, Alison Babeu, Lisa Cerrato, Anna Krohn, Frederik Baumgart, Monica Berti, Greta Franzini, and Simona Stoyanova. 2014. "Cataloging for a Billion Word Library of Greek and Latin." In *DATeCH '14: Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*, 83–88. New York: ACM. doi:10.1145/2595188.2595190.

Crane, Gregory, Brent Seales, and Melissa Terras. 2009. "Cyberinfrastructure for Classical Philology." *Digital Humanities Quarterly* 3 (1). http://www.digitalhumanities.org/dhq/vol/003/1/000023/000023.html.

Del Grosso, Angelo Mario, and Federico Boschetti. 2013. "Collaborative Multimedia Platform for Computational Philology: CoPhi Architecture." In *MMEDIA 2013, The Fifth International Conferences on Advances in Multimedia* [Proceedings], edited by Philip Davies and David Newell, 46–51. N.p.: IARIA. http://www.thinkmind.org/index.php?view=article&articleid=mmedia_2013_3_10_40059.

Di Iorio, Angelo, Michele Schirinzi, Fabio Vitali, and Carlo Marchetti. 2009. "A Natural and Multi-layered Approach to Detect Changes in Tree-based Textual Documents." In *Enterprise Information Systems: Proceedings of 11th International Conference on Enterprise Information Systems (ICEIS 2009)*, edited by Joaquim Filipe and José Cordeiro, 90–101. Lecture Notes in Business Information Processing 24. Berlin: Springer. doi:10.1007/978-3-642-01347-8_8.

Fowler, Martin. 1996. *Analysis Patterns: Reusable Object Models*. Menlo Park, CA: Addison Wesley.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.

Hohpe, Gregor, and Bobby Woolf. 2004. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston: Addison-Wesley.

Lamé, Marion. 2012. "Epigraphie en réseau: réflexions sur les potentialités d'innovations dans la représentation numérique d'inscriptions complexes." Ph.D. thesis, University of Bologna – University of Aix-Marseille. http://www.theses.fr/2012AIXM3130.

Martin, Robert C. 2000. "Design Principles and Design Patterns." Gurnee, IL: Object Mentor. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf.

Peroni, Silvio, Francesco Poggi, and Fabio Vitali. 2013. "Tracking Changes through EARMARK: A Theoretical Perspective and an Implementation." In *DChanges 2013: Proceedings of the International Workshop on Document Changes: Modeling, Detection, Storage and Visualization*, edited by Gioele Barabucci, Uwe M. Borghoff, Angelo Di Iorio, and Sonja Maier. Aachen, Germany: CEUR Workshop Proceedings. http://ceur-ws.org/Vol-1008/paper6.pdf.

Pierazzo, Elena. 2015. *Digital Scholarly Editing: Theories, Models and Methods*. Farnham, Surrey: Ashgate.

Ronnau, Sebastian, Geraint Philipp, and Uwe M. Borgho. 2009. "Efficient Change Control of XML Documents." In *DocEng 2009: Proceedings of the 9th ACM Symposium on Document Engineering*, 3–12. New York: ACM. doi:10.1145/1600193.1600197.

Rosenberg, Doug, and Matt Stephens. 2007. *Use Case Driven Object Modeling with UML: Theory and Practice*. Berkeley, CA: Apress.

Schmidt, Desmond. 2014. "Towards an Interoperable Digital Scholarly Edition." *Journal of the Text Encoding Initiative* 7 (November). http://jtei.revues.org/979.

Schmidt, Desmond, and Robert Colomb. 2009. "A Data Structure for Representing Multi-version Texts Online." *International Journal of Human-Computer Studies* 67 (6): 497–514. doi:10.1016/j.ijhcs.2009.02.001.

Siemens, Ray, Meagan Timney, Cara Leitch, Corina Koolen, and Alex Garnett. 2012. "Toward Modeling the *Social* Edition: An Approach to Understanding the Electronic Scholarly Edition in the Context of New and Emerging Social Media." *Literary and Linguistic Computing* 27 (4): 445–61. doi:10.1093/llc/fqs013.

TEI Consortium. 2015. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Version 2.8.0. Last updated April 6. N.p.: TEI Consortium. http://www.tei-c.org/Vault/P5/2.8.0/doc/tei-p5-doc/en/html/.

Terras, Melissa, and Gregory Crane, eds. 2010. *Changing the Center of Gravity: Transforming Classical Studies through Cyberinfrastructure.* Digital Technologies and the Ancient World 4. Piscataway, NJ: Gorgias Press.

Wittern, Christian, Arianna Ciula, and Conal Tuohy. 2009. "The Making of TEI P5." *Literary and Linguistic Computing* 24 (3): 281–96. doi:10.1093/llc/fqp017.

NOTES

1 The ODD files currently available can be downloaded from GitHub: https://github.com/CoPhi. The TEI schema we intend eventually to adopt conforms to the EpiDoc vocabulary, following the policy of the Perseus Catalog (Crane et al. 2014).

2 Facelets XML templates are used under the Java Server Faces 2.0 specification.

3 http://www.w3.org/standards/xml/.

4 http://cocoon.apache.org/.

5 http://exist-db.org/.

6 http://www.tustep.uni-tuebingen.de/tustep_eng.html.

7 http://dcl.ils.indiana.edu/.

8 http://sourceforge.net/projects/txm/.

9 http://tapasproject.org/.

10 See the Agile Manifesto (http://www.agilemanifesto.org/) for a detailed explanation.

ABSTRACT

In this article we illustrate a work in progress related to the design of a library of software components devoted to editing, processing, and visualizing TEI-annotated documents in the domain of philological studies, in particular in the subdomain of collaborative philology, which concerns the social activity of scholars focused on shared philological tasks. We discuss the technologies related to XML markup languages and the processing of marked-up documents. We describe the method used to design and implement the TeiCoPhiLib, outlining the design patterns as well as discussing general benefits of the overall architecture. Finally, we present case studies in which some components of our library currently implemented in Java have been used.

INDEX

Keywords: APIs, design patterns, library of components, collaborative philology

AUTHORS

FEDERICO BOSCHETTI

Federico Boschetti, holds a PhD in classical philology and brain and cognitive sciences—language, interaction and computation. He is a researcher at the A. Zampolli Institute for Computational Linguistics of the Italian National Research Council (ILC-CNR) of Pisa. His area of interest is computational and collaborative philology.

ANGELO MARIO DEL GROSSO

Angelo Mario Del Grosso, holds a PhD in information engineering at the Engineering Ph.D. School "Leonardo da Vinci" of the University of Pisa. He is a research fellow at the A. Zampolli Institute for Computational Linguistics of the Italian National Research Council (ILC-CNR) of Pisa. His area of interest is the analysis, design and development of object-oriented and computational methodologies applied to the textual scholarship domain.