UNIVERSITÀ DI PISA
Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
Ingegneria dell'Informazione

Tesi di Dottorato di Ricerca

Designing a Library of Components for Textual Scholarship

Angelo Mario Del Grosso

Anno 2015

Università di Pisa
Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in Ingegneria dell'Informazione

Ph.D. Thesis

Designing a Library of Components for Textual Scholarship

Candidate:

Angelo Mario Del Grosso

Supervisors:

Prof. Francesco Marcelloni

Dr. Andrea Bozzi

Dr. Federico Boschetti

Dr. Emiliano Giovannetti

2015

SSD ING-INF/05

Sommario

Il presente lavoro affronta e descrive temi legati all'applicazione di nuove tecnologie, di metodologie informatiche e di progettazione software volti allo sviluppo di strumenti innovativi per le Digital Humanities (DH), un'area di studio caratterizzata da una forte interdisciplinarità e da una continua evoluzione. In particolare, questo contributo definisce alcuni specifici requisiti relativi al dominio del Literary Computing e al settore del Digital Textual Scholarship. Conseguentemente, il contesto principale di elaborazione tratta documenti scritti in latino, greco e arabo, nonché testi in lingue moderne contenenti temi storici e filologici. L'attività di ricerca si concentra sulla progettazione di una libreria modulare (TSLib) in grado di operare su fonti ad elevato valore culturale, al fine di editarle, elaborarle, confrontarle, analizzarle, visualizzarle e ricercarle. La tesi si articola in cinque capitoli. Il capitolo 1 riassume il contesto del dominio applicativo e fornisce un quadro generale degli obiettivi e dei benefici della ricerca. Il capitolo 2 illustra alcuni importanti lavori e iniziative analoghe, insieme a una breve panoramica dei risultati più significativi ottenuti nel settore delle DH. Il capitolo 3 ripercorre accuratamente e motiva il processo di progettazione messo a punto. Esso inizia con la descrizione dei principi tecnici adottati e mostra come essi vengono applicati al dominio d'interesse. Il capitolo continua definendo i requisiti, l'architettura e il modello del metodo proposto. Sono così evidenziati e discussi gli aspetti concernenti i design patterns e la progettazione delle Application Programming Interfaces (APIs). La parte finale del lavoro (capitolo 4) illustra i risultati ottenuti da concreti progetti di ricerca che, da un lato, hanno contribuito alla progettazione della libreria e, dall'altro, hanno avuto modo di sfruttarne gli sviluppi. Sono stati quindi discussi diversi temi: (a) l'acquisizione e la codifica del testo, (b) l'allineamento e la gestione delle varianti testuali, (c) le annotazioni multilivello. La tesi si conclude con alcune riflessioni e considerazioni indicando anche possibili percorsi d'indagine futuri (capitolo 5).

Abstract

The present work is the result of the research activity carried out during my PhD studies. This thesis addresses the application of new technologies, computer science methodologies, and software design principles in the interdisciplinary and evolving field of DH - in other contexts known as Humanities Computing. In particular, this contribution highlights the specific needs entailed in collaborative literary computing and in digital textual scholarship. The source context especially concentrates on documents written in Latin, Greek and Arabic, or on documents in modern languages concerning historical and philological topics. In the specific, the research activity focuses on the design of a modular library (TSLib) dealing with scholarly sources for what regards their editing, processing, comparison, analysis, visualization and searching. The thesis explores the aforementioned topics across five chapters. Chapter 1 tracks the context of the digital textual scholarship and gives a summary of the objectives and the benefits of this research. Chapter 2 illustrates related works and similar initiatives, along with worth mentioning projects and outcomes in the area of Digital Humanities. Chapter 3 thoroughly describes and motivates the design process implemented. The process starts by describing well-known engineering principles and shows how they are applied for the digital textual domain and for the computational scholarly needs. Then, it continues introducing requirements, architecture and models of the proposed method. Design issues with regards to patterns and APIs are highlighted. The final part of this work (chapter 4) illustrates concrete results deriving from a number of research projects that, on the one hand, have contributed to the design of the library and, on the other hand, have based their work on it. Several topics have been discussed: (a) acquisition and text encoding, (b) alignment and variant annotation, and (c) multi-level annotation. In the conclusion, a few reflections and considerations are presented, together for suggestions and for further studies (chapter 5).

<dedication>	→	*<people>*
<people>	→	*<those I love>*
<those I love>	→	*<name> <surname>*

(after Thomas A. Sudkamp)

Contents

List of Figures

Chapter 1

Introduction

This chapter is divided in two main parts. The first part aims at providing a brief historical overview of the phases which textual scholarship has gone through. The second part presents the relevance, motivations, objectives and challenges that have characterized this research work. The main focus is placed on the advantages of designing a library of components for textual scholarship.

1.1 Overview

The digital revolution and the widespread availability of documents through the World Wide Web have increasingly changed the ways of studying and knowing textual resources and cultural heritage documents [1, 2]. New technologies offer a mean to encode, process, and link relevant textual-phenomena. This offers the opportunity to deal with an amount of data and automation that would be impossible with traditional methodologies. Moreover, this process has played an important role in expanding the access to textual resources for studies and for digital scholarly editing[1] [6, 7]. As a consequence, computer science has extended the scope of its research to matters related to the field of humanities [8, 9]. Following this, computer science and computer engineering have also started to deal with interdisciplinary issues trying to solve literary problems in an effective and efficient way [10, 11].

As claimed by I. Lancashire [12]:

[1]Digital scholarly editing covers the whole editorial, scientific and critical process that leads to the publication of an electronic resource [3, 4, 5].

> Literary and linguistic computing in the humanities analyzes texts to give insight into their meaning, their authorship, and their textual tradition. These insights are derived from computational text analysis, which employs concordancers to find verbal patterns. These can sometimes help uniquely identify the author's idiolect, illuminate the chronology, narrative structure, semantics, and sources of a text, and create a hypothetical stemma representing the descent of the text from earlier versions. Norbert Wiener's cybernetics, a theory of messages that applies equally to communications by human beings and machines, underpins the computational study of author, text, readers, and the history of textual transmission.

At the time of writing, the field of DH[2] is going through a crucial methodological debate. The discussion addresses both its definition and its main objectives [13, 14, 15, 16, 17, 18].

For the purpose of this thesis, a brief overview of the historical milestones that have characterized it might be necessary. In particular, a clarification is needed regarding digital textual scholarship[3] and computational philology[4]. This improves the understanding of the context in which this study has been carried out.

The first project commonly acknowledged to belong to the field of Digital Humanities dates back to 1946. With the help of IBM, Roberto Busa began to create a digital corpus composed of eleven million words, based on the works of Thomas Aquinas (Index Thomisticus) [20, 21].

For a long time, the principal subject of investigation had been the process of data acquisition and the long-term preservation of documents in trusted repositories [22]. Consequently, digital and computational humanists highly depended upon the digitization of primary sources. Digitizing source images was the first

[2]The Manifesto for the Digital Humanities is avaiable at the following URL: http://tcp.hypotheses.org/411

[3] As pointed out in [7, 19], digital textual scholarship deals with studying by means of a digital environment all and each aspect of texts in order to understand their sources and history. It encompasses any form of investigation of original documents, whether manuscript or print, to establish a work's composition, revision, editing, printing, production, or distribution.

[4]Computational philology refers to a set of activities related to the development of programs which are devoted to encourage interaction between the publisher and the machine [3]. The expression *computational philology* has been used for the first time in 1968 inside a computer science document about computational linguistic, see S. Kuno - A.G. Oettinger, "Computational Linguistics in a Ph.D. Computer Science Program", Comm. ACM 11, 12 (1968), in part. p. 835.

step for any initiative of this kind. Conversely, few studies were conducted on digital editions and almost none on their publication [23].

In the beginning, digitization was typewritten and individually corrected by hand. At that time, scholars in the field of humanities had limited technological experience in word processors. In addition the workflow of the digitization mixed automatic process and human adjustment with consequent issues related to scalability. Scholars' habits directly derived from paper libraries and they extensively imitated printed forms [24].

In order to increase the amount of textual collections, in the early 70s, community-driven projects were started, for instance the Gutenberg Project. Their main aim was to create large digital archives containing the works of a number of authors. However, the quality of these early digitization initiatives was not adequate to scholars' needs.

Two main issues are worth mentioning: 1) the impossibility to compare the image of the original printed edition with the digital text derived from it; 2) the absence of any para-textual data, such as the preface, the introduction or the footnotes, as well as data for editorial purposes (i.e. availability of critical apparatus information stored in a database [25]). Later on, digitization campaigns were also extended to historical documents and particularly to literary corpora of Greek and Latin texts. At that time, digital resources used CD-ROM storage technology to preserve and disseminate the content of the archives. However, communication between humanists and computer scientists was uncommon, except for a few pioneers[5] with a wider vision who mediated among the members of these communities. For instance, some best-known works explored computer programs for the creation of machine-readable representation of literary resources [26, 27]. The statistical methods applied helped identifying the genealogical relationship among several codes [28, 29], and analyzing the results of the collation phase [30].

Afterwards, standards for encoding were designed that allowed interoperability between different textual representations. At character level, one of the first achievements was the reduction of encodings for glyphs not directly mapped in the ASCII table. Related to that, the creation of the BetaCode scheme [31] was one of the most adopted solutions, especially for ancient languages. At document level, public Document Type Definitions (DTDs) achieved easier and long-term preservation of individual documents and collections. From that moment on, scholars

[5]Richard Goulet, Peter Robinson, Andrea Bozzi, and Gian Piero Zarri, among others, are researchers who first investigated the use of the computer in philology.

started using digital methods for the textual editing. Nevertheless, they continued printing documents, due to the impossibility of publishing them in a digital form. In the same period editorial programs emerged, like TUSTEP [32], or many others based on the TeX family [33, 34] .

In the 90s, other initiatives came forward which aimed at storing document images alongside full texts. Applications met at least two requirements. Firstly, they linked textual chunks to relative Region of Interest (ROI), automatically or semi-automatically; secondly, they provided users with the page image up front, with a full text search capability and with annotation tools. This way, users were facilitated in solving problems of interpretation caused to difficult reading due to damages to the physical support [35].

Contextually, digital libraries and philological environments began to include links to the primary sources of each information. In such a way, readers were able to compare editorial choices across multiple sources enabling editorial discussions [22].

In this framework, digital scholars worked for marking up documents which were made up of two fundamental parts: a) the textual content, and b) the metadata. However, computer authoring systems supported only the encoding phase, not allowing any data processing. Although this approach replaced printed books with new digital editions, it did not foster a real change in the scholar paradigm [24]. Moreover, software applications aimed at solving the needs of specific projects, often disregarding reusability and extensibility [36, 10, 37]. In light of this, scholars have been working together with ITC companies in order to experiment and develop new computer-aided methods. Among others, tools were able to manage quotations and variant readings, to link to pre-existing bibliographic documents, to activate services such as linguistic analysis, or to mark locations on a map [4, 25, 38].

The possibility of disseminate electronic editions exploited infrastructures and applications like Cocoon [39] and Anastasia [40]. As a consequence, companies with technological know-how, often used to make business by selling the implemented platforms.

More recently, digital repositories have been managing massively cultural heritage materials both by using multiple OCR engines and by processing linguistic data. This takes place through NLP tools and text alignment procedures. In addition this technological improvement is mainly due to three aspects: (a) the wide diffusion of Internet (particularly the Word Wide Web service), (b) the development and adoption of machine learning algorithms, and (c) the successful outcomes achieved in automatic detection of genetic variations. At this stage, digital platforms offer a new space where scholars can sign up in order to proofread the content of the archive and to open collaborative intellectual comments [41]. Collaborative philology has begun to take hold: users, by means of a suitable cyber-infrastructure, evaluate automated processed data which has been statistically classified.

This new approach has led to a paradigm shift in a threefold way:

1. Systems based on stochastic models and statistical algorithms started being employed by crowd-sourcing communities. Up to that time, automatic systems had made use of procedures based on a set of rules which could handle, in an asymptotic manner, all possible cases of input data.

2. Experiments of collaborative textual criticism has fostered theoretical considerations on the digital representation of the text (decentralized, real-time community contributions, dynamic and fluid representation)[3, 42].

3. Digital libraries have become capable of scanning their contents and searching for new data such as secondary sources and new editions of primary materials.

Finally, since 2000s, researchers have increasingly gained expertise in both computational and textual domains. They have put forward new issues, mainly seeking methods to be as open and share as possible. For instance, scholars claim open data access and source code availability. On the one hand, they can develop tools and implement experiments based on reusable components. On the other, they can collaborate directly with the community for bug fixing and for extending the functionality. In fact, experts have an important part to play both for the scholarly editing and for the software development.

1.2 Motivation, goals, and challenges

The objective of this work is to discuss modeling issues and prototype applications in designing a modular software library for the textual scholarly domain. It tries to highlight and address the technological gap existing between digital scholars approaches and those approaches used by software engineers and computer scientists.

Up to now, new technologies and digital methodologies are common in many areas of research such as natural sciences, computational linguistics or bio-informatics. Unfortunately, they are not yet mature for the literary computing field and particularly for computational philology. Nevertheless, researches in such a domain are constantly shifting classical practices towards various forms of digital representations and processing of textual resources and their transmission as historical process. As a matter of fact, the digital environment allows textual researchers to implement methods, approaches and tools in order to:

- browsing in large archives of electronic texts and digital images;
- promoting the discovery of textual, para-textual, extra-textual, and inter-textual phenomena (i.e. parallel passages, citations, annotations, etc.);
- detecting errors and variant readings where texts are transmitted by multiple witnesses;
- supporting the editing of a critical apparatus and consequently the editing of a critical edition;
- producing indexes and concordances both from texts and from scholars comments;
- improving the ability to make correct editorial choices;
- working on a text collaboratively, with multiple users, even those coming from different areas of specialization (philologists, linguists, historians, philosophers, etc.).

Generally, this kind of studies deals with the following complementary points: a) producing accurate and detailed digital resources (digital philology); b) developing sound, efficient and flexible software to process resources (computational philology); and c) integrating data and services in a virtual research infrastructure (ePhilology).

a) Digital philology focuses on the acquisition and the creation of digital resources. For instance, improving OCR accuracy applied to classical and less-served languages, or establishing standards for shared annotations.

b) Computational philology, instead, concerns data analysis and data manipulation. For example, the development of procedures for word frequencies computation, as well as the comparison of two or more texts, the automated lemmatization or the linguistic analysis of texts.

c) Finally, e-philology aims at building research infrastructures in order to handle collaboration among scholars (e.g. social annotations or social editions), human-machine interaction (e.g. by using search engines and visualization techniques), and machine to machine cooperation (e.g. by software agents, which use chains of web services or linked open data).

The above points actually provide the environment for the development of tools that aid literary studies in the digital age. In turn, literary studies involve four fundamental aspects as regards the digital representation of textual sources:

- multiple devices convey the textual content of the source (e.g. the text of a manuscript and the image of a page);
- the tradition of the text manifests multiple variant readings (e.g. the word disagreements found in the same textual context within different document-witnesses);
- the textual content encompasses multiple layers of analysis at different levels of granularity (e.g. the syntactic analysis of a sentence or a metrical analysis of a poetic verse);
- the levels of analysis provide multiple interpretations (e.g. the disagreements concerning morphological interpretations based on the same word).

The aforementioned accounts show that the challenge of this research concerns the modeling of data in order to serve as evidence for literary studies in digital ecosystems. In fact, in the last few decades, much work for textual scholarship has leveraged the idea that the design and the development of digital tools could be carried out by reusing and customizing the outcomes obtained in other computational sciences (e.g. natural language processing, bioinformatics, and others).

However, the tradition of scholarly studies needs digital models and computational approaches which rely on appropriate data structures, algorithms and functionalities. For example, computational linguistics analyzes a single text flow associated to single linguistic analyses (e.g. syntactic and semantic analyses), whereas, for example, computational philology must deal with multiple versions of the same text (due to variants in the manuscripts or conjectural emendations provided by the scholars) and multiple interpretations at each level of analysis (due to the disagreement of authoritative scholars recorded in several commentaries across the centuries).

Consequently, textual scholarship models extend those models commonly adopted for the linguistic software design. In particular, the basic model takes into account four core properties:

1. the version of the textual data;
2. the analysis and interpretation of the information;
3. the granularity of the analysis;
4. the location of the textual data.

In this background, a suitable textual scholarship application, which deeply exploits the possibilities of digital technology, allows, in a fully integrated virtual environment, the interconnection among primary sources (such as images or diplomatic/interpretative transcriptions of a manuscript), the shared access to secondary materials (such as commentaries, monographs, etc.), and the availability of scholarly tools (such as collection modules, concordances, etc.). In such a way, a digital edition provides scholars with a powerful tool which allows an effective and efficient document dissemination among literary communities.

This work, therefore, is an insight into the design of software components focusing on literary activities. These activities cover, for example, the alignment of complex textual objects (e.g. the alignment of variant readings according to the edit distance of the inflected forms and to their semantic similarity), the management of levels of analysis (e.g. morphological or lexical analysis by using different methods such as statistical algorithms or rule-based approaches), the editing and retrieval of multiple, concurrent annotations. Furthermore, another important functionality is the linkage of textual resources to multimedia sources, for example the image of the manuscript page with its textual content. Three main building blocks are involved for developing a framework able to deal with such wide needs:

1. Acquisition of resources by Optical Character Recognition (OCR) or information extraction and document transformation from semi-structured resources to structured ones (ETL).

2. Document and text processing, content analysis, data indexing and information retrieval.

3. Development of collaborative environments based on well-defined components which provide interfaces for developers (API), for service provides (SPI), and for users (GUI).

Eventually, the research outcomes relate to the design of components, of modules and of plug-ins for a collaborative object-oriented library. The system establishes a suitable tool, on the one hand, for analyzing manuscripts and printed documents and, on the other hand, for producing new critical editions. For this purpose this research work has investigated how to design a worthwhile API for large-scale and long-term scholarly projects.

The method adopted for analyzing, designing and implementing the artifacts uses the following approaches: a custom agile use-case driven process, the object-oriented paradigm, relevant design principles, and the unified modeling language diagrams (UML). Doing so, research studies can draw up different views of the domain under study focusing on both static and dynamic aspects of the system. The UML graphical notation is important as it helps in designing a model at high level of abstraction. In this way, the model is platform and technology independent. Due to the abstract modeling and the modular design, the concepts within this research have already been used in a number of national and international projects devoted to manage parallel multimedia resources, both text and image.

At present, DH engages a large and heterogeneous community. A rich set of methodologies, approaches, procedures, tools, and applications have been developed within it. Unfortunately, the outcomes are frequently isolated and kept in unshared black-boxes. Furthermore, design features such as interoperability, reusability and extensibility, have hardly been taken into account. This is mainly due to the fact that a great number of DH researchers use private policies in developing new models for their computational studies [43]. Indeed, current software systems for academic studies generally depend on single initiatives and language requirements; thereby, each language requires a different system for language analysis. Contrarily, computational projects involving cultural heritage materials cannot be addressed through specific needs or languages. Even less they cannot rely on rigid and monolithic applications [44, 45, 7, 46]. As a result, one of the great objectives of DH is to design shared methods, efficient algorithms and reusable libraries which can effectively meet and solve a wide variety of textual requirements [47].

Hence, a crucial and expected accomplishment is to empower the community of textual studies through an integrated service-driven framework tailored to their specific needs. Therefore it is necessary that software engineers, computer scientists and computational humanists combine their efforts in order to develop flexible, reusable and reliable technologies for the domain of social sciences and humanities (SSH). In other words, the main objective lies in designing and implementing a framework able to assemble software packages (components) to be exploited by computational inquiries in the field of humanities. Afterwards, software artifacts have to be integrated into a comprehensive technology platform which enables computational scholars to build new tools or to refactor existing ones. This ongoing process provides scholars the necessary knowledge for the creation of complex and reliable software, thus supporting the DH field. Consequently, a modular library with well-designed Application Programming Interfaces is now deemed strategic for conducting research in the area of computational philology. Several large-scale researches have recently been released to the community with regards to these issues (see chapter 2).

This kind of infrastructure, on the one hand, provides classical scholars with a digital environment for studying textual documents. On the other hand, it serves as a platform for computational scholars for developing and sharing their own software applications.

Nowadays, applications for textual scholarship are becoming increasingly complex. Therefore, it is necessary to apply the best engineering practices in order to build them properly and make them reliable. According to the aforementioned framework, the aim of this research is to model and implement flexible and reusable software components tailored on the requirements of the computational and collaborative textual scholarship. For example, such components must allow their users to detect the resource language, choose a suitable morphological engine, perform analyses at different granularities, manage text repository, and construct training sets for further statistical analysis.

In order to fulfill these goals, three points need to be particularly stressed:

1. Software engineering principles;

2. Literary computing applications;

3. Linguistic technology methods.

The first point involves: (a) adopting a process which considers Agile principles, user requirements and domain modeling; (b) fostering the object-oriented paradigm; (c) following the component-based model; (d) considering well-known design patterns; (e) putting into effect practical API design techniques; (f) refactoring existing software; (g) establishing integration policies; (h) favoring the open-source community-based development; (i) producing a test case and unit test for the artifacts.

The second point involves: (a) implementing efficient methods for content acquisition such as optical character recognition, (b) developing reliable document processing algorithms and scholarly editing systems such as software that handles variant readings and quotations, (c) formalizing shared data-format schemes and common ontologies such as mark-up XML vocabularies and formal domain conceptualizations.

Finally, the third pillar encompasses, among others, (a) implementing Natural Language Processing (NLP) tasks such as lemmatization, parsing, classification, fuzzy matching, searching, etc.; (b) enhancing accuracy for current algorithms such as improving recall and precision in classification or clustering processes; (c) improving scalability of the current approaches such as parallelizing tasks for massive digitization, or developing language-independent strategies; (d) researching new strategies, algorithms and data structures focusing on complex matters; (e) providing useful visualization like an immediate view of the content of a document alongside the linguistic analysis or a suitable view of the textual phenomena with the related image area.

Generally, design principles help to use the full power of object-oriented programming. Additionally, where applied, they enable to write efficient, effective, flexible, extensible, scalable and maintainable software. This deals with more focused and manageable software modules and source code. As a consequence, developers can concentrate on the actual logical needs of the application or on extending the software already available. It is worth noting that without an effective engineering approach within the DH applications, the cost of software development would be higher than necessary, and the cost of maintaining and using the tools would be unfeasible.

Sharing computational tools and technologies throughout common frameworks is rather a new theme for literary scientists. Indeed, up to now collaborations and data exchanges have been defined in terms of data format or data encoding, or at most exposing web services. However, at moment digital scholars are ready for developing software in distributed teams and classical philologists are further studying in virtual environments. It is therefore necessary to create a new generation of Citizen Scientists [45].

In conclusion, scholars, working together with computational researchers, could make textual heritage sources more accessible and intelligible. Moreover, research in Digital Humanities demands new technologies in order to face the following challenges:

Firtly, researchers should try to establish a process for creating and sharing data and tools for the target field. Consequently, research can address the design and the development of the desired applications applying scalable methods with which managing the digital sources available for analysis. Afterwards, applications can focus on providing better Optical Character Recognition (OCR) for improve digital document processing. Moreover, it is fundamental to provide alignment and collation mechanisms of textual readings by exploiting parallel and distributed environment.

Secondly, researches should manage the growing collection of encoded data such as morpho-syntactic annotations, named entity data, and information about interrelated corpora. Therefore, scholarly tools have to support a diverse range of annotations, such as translations, studies of textual transmission, quotations, prosopography, and linguistic information (semantic and lexical analyses), thereby managing the circulation of ideas across time, space, language and culture. In particular digital scholarly applications need to support facilities for scholarly editing and for reconstructing texts [48]. Specifically, such a feature not only involves the management of variant readings but also the document processing pertaining to stemmatology matters (i.e. by phylogenetic methods) [49]. Obviously, scholars require tools to enhance the ability of researchers to work with historical sources.

Thirdly, an other important area of study involves the implementation of search engines which consider textual variations. This presumes that repositories dispose of multiple editions as well as tools that manage different versions of the same text. These search engines need to collate multiple editions against each other, identifying quotations of primary sources and providing differences. Moreover, they should give access to repositories of conjectures, to bibliographical documents, to high resolution images, to commentaries and to all other information that scholars may need.

Finally, one of the objectives of the common work among textual scholars, software engineers and computer scientists is the setting up of a virtual textual research environment able to:

- implement methods for crowd-sourcing and citizen science support;
- provide integrated access to digital research resources, to tools and services across disciplines and user communities;
- process structured and qualitative data in virtual workspaces;
- handle textual contents in several languages and localize the environment into multiple languages by translating the core vocabularies;
- promote digital practices for research, education and innovation, sharing material for further computational processing;
- allow young scholars to interact primary sources and to browse the vocabulary, morphology and syntax of the texts they are reading;
- preserve continuity for textual information and textual tradition;
- define, semantics and ontologies concerning metadata, as well as, guarantee the best levels of abstraction to process data in order to ensure interoperability;
- promote effective collaboration among researchers and simplify access to data discovery and its use.

To sum up, the textual scholarship field has been moving towards computational research, despite the considerable delay if compared to other scientific disciplines. Indeed, in the digital era it is necessary to implement tools that allow scholars to handle fluid digital editions dynamically [42]. This follows that each text tradition generates multiple variant readings, levels of annotations and interpretations. Consequently, computational applications are fundamental instruments that support editorial process. It is worth noting that, although new technologies improve the work of scholars, only philologists have the authority to establish the text (e.g. making the editorial choices). In view of what is now possible, this thesis argues that the primary tools of textual scholarship should become software components able to process digital editions in a collaborative way.

1.3 The benefits of a library of components

As pointed out by [11, 19, 13, 50, 7], in the domain of literary computing, many existing software models and their implementations are not modular, customizable and do not scale. Consequently, they do not encourage reusability and flexibility of the artifacts. Nowadays, software applications, which need distributed and interrelated functionalities, integrate outcomes and capabilities by means of data exchange and web services. Although effective and well tested, this approach does not encourage a full model sharing, a community-based development, and service reliability.

For this reason there is a claimed need [51] of shared tools for digital scholarly editing as well as for literary material processing. This research attempts to offer a solution to fill this gap. This study starts from the assumption that a software library provides opportunities for (a) designing suitable use cases, (b) defining domain entities and abstract data types, (c) modeling functionalities and behavior, (d) organizing data, (e) exporting services. Therefore, computing applications hold the potential to reuse components and access or manipulate data in an efficient and shared way. This process is paramount to address requirements in the field of DH and it provides scholars with the appropriate services to create complex and trustable software.

A well-designed library should bear the following general features:

- flexibility and modular application architecture;
- platform independence;
- customizability to the end user needs;
- ability to work online as well as offline;
- simplified distribution to the end user;
- simplified updating for the clients;
- consistency that avoids unexpected outcomes for users;
- ability to evolve while maintaining backward compatibility;
- collaboration among members of the community;
- cooperation among services;

- stable contract that can be used for communication purposes.

Specifically, a software library offers entry points on top of which the user can assemble related parts of the application.

Furthermore, component-based libraries provide strategies for customizing applications by means of extensible modules and plug-ins. As an outcome, the systems seem to have been created as a whole. In addition, a modular library is composed of smaller, separate units, which are well isolated and uniquely identified. These units export simplified interfaces and describe the environment they need in order to work properly. This allows the core components of the library and the final applications to assemble the module as required. Suitable Application Programming Interfaces identify services and data types of long-term value and make them accessible to the target community with more stability. Finally, components can be integrated into existing projects and adapted without further modifications to their source code. This also allows avoiding low-level programming interventions. In such a way, a fully functional software package can be created addressing the needs of the domain community. This work begins by defining a proof-of-concept library for literary studies and several API principles that provide stable access to the library functionality. API guarantees a set of component services able to handle textual resources for historical documents in the domain of literary computing. Through this process, the library provides functions that support scholarly works, such as aligning complex textual objects, expanding levels of analysis, editing and retrieving multiple and concurrent annotations, processing manuscript page images.

Each component is interoperable with other components and, at the same time, it abstracts itself from the underlying technology and implementation. The component abstraction is made possible by designing standard building blocks, shared workflows and well documented interfaces with coherent APIs. Application Programming Interfaces provide abstractions for problems and they specify how clients should interact with software components that implement a solution to that problem. In essence, APIs define reusable building blocks that allow modular pieces of functionality to be incorporated into end-user applications as loosely coupled components. Designing proper APIs is strategic mainly for two reasons: 1) the long-term availability of the library functionality and 2) the community-driven development of the tools. Moreover, the proper design results in reusable software artifacts that reflect the target domain. Although the aforementioned approach is well-known by researchers in science and technology, it is not a common practice in humanities computing.

Users benefit from a library as it has to clearly distinguish between the specification and implementation of Abstract Data Types (ADTs) [52]. Moreover, scholars can benefit from the Textual Scholarship Library (TSLib) because it is being designed and tested adopting use-case scenarios gathered from concrete research projects. By adopting API principles, libraries export services through a logical interface and hide the details underpinning the internal implementation. Therefore, textual scholarship API can offer a high-level of abstraction for scholar needs and promotes code reuse by sharing the same functionality thus allowing, at the same time, multiple service providers.

To conclude, a modular and abstract library would have a twofold result. In the first place, it would allow developers to reuse the general model and its core components; secondly, develpers would be able to add functionalities and solve new requirements by plugging extensions into the system [46]. Therefore, Application Programming Interfaces offer a means of access to ad-hoc implementations tailored on specific services -such as linguistic morphological analysis-, through a specified engine.

Chapter 2

Background

Due to the wide range of themes covered by Digital Humanities and its interdisciplinary nature, this field is in constant evolution. Nevertheless, this chapter seeks to take a snapshot of the state of the art in the application of computational studies to textual scholarship. Moreover, it tries to give an overview of the context in which this research is placed and the theories and technologies supporting them. Furthermore, the chapter introduces the most outstanding software libraries, nearly all implemented in Java, and used for research purposes.

2.1 Preliminary remarks

Engineers, computer scientists, philologists and linguists have given impulse to a wide research activities and to many initiatives in the area of Digital Humanities (DH). They have worked together for more than thirty years to implement automatic systems applied to human sciences. Hence, experiments, new approaches, theories, techniques and applications have evolved in relation to the handling and automatic management of texts and languages. Collaborations, conventions, international and interdisciplinary associations have arisen with the aim of guiding the development of researches.

Text editing is one of the activities that has benefited most from the spread of computers. For example, typos can be kept under control thanks to spell-checkers, documents can be saved in multiple copies and easily sent via e-mail or, more recently, stored and shared in the *cloud* [53].

Particularly in the case of scholarly editing, the availability of digital documents, either in form of digital texts or scanned manuscripts, can considerably simplify philologists' work. An evidence is given by the possibility of accessing resources through searching a particular word of interest. The importance of this area of research is also confirmed by the many initiatives and tools in the fields of literary and linguistic computing and of natural language processing for cultural heritage [54, 55].

Moreover, the possibility for a document to be edited by a variety of people has gained increasing interest. A related practice, the so-called *Collaborative Editing*, allows scholarly communities produce *social* works giving to the members the possibility of contributing to establish the texts [1]. One of the key features of collaborative editing is its asynchronicity, which allows a dramatic reduction of work time. Collaborative editing constitutes a particularly useful instrument in scholarly editing, as it envisions the Web as a common workspace. The rapid increase of wiki projects and the wide use of Google Documents (GDocs) are a result of this vision. Wiki is a website characterized by (mostly textual) contents that multiple users can edit in a collaborative way, by using a simple browser. It also disposes of a system that ensures tracking of the changes brought to documents. However, this platform presents some limitations. Firstly, it does not support either advanced search engines, nor versioning of dependencies. Consequently, the references reported in each document version are always linked to the most up to date documents. GDocs environment, on the other hand, is the most used collaborative textual editing tool of general-purpose documents. Differently from wikis, GDocs is a real-time collaborative editing framework, allowing users to edit and to comment documents simultaneously. Both technologies represent a model for developing systems for digital scholarly editing. As witnessed by the aforementioned tools, current trends in digital and computational philology are oriented towards developing collaborative environments [56, 44, 41]. Research centers are developing technologies and methods in order to provide the international community with suitable tools to make, share and to reuse textual scholarship products. This would also widen the access of scholars to gather and edit material [1, 36]. Several international congresses recently argued about this topic [57, 58, 59, 60]. Issues related to licenses play a crucial role in this context[1].

[1]This work does not take in consideration licenses. However, as regards data, Creative Commons is the mean to deal with these topics. As far as software, Open Source Initiative (OSI) and Free Software Foundation (FSF) are the organizations which manage such a matter.

2.2 Initiatives for textual scholarship

In recent years, relevant projects have been conducted on textual scholarship, which reflect the objectives and trends discussed so far. The majority of these researches suggest solutions to implement digital libraries for general purposes, or mono-thematic archives. Normally, they focus on Web access to repositories, both of texts and images available through interconnected frameworks [44]. This section outlines the initiatives that are most representative in the context of this thesis, since they address the field of computational and digital philology. The projects are grouped into three classes, namely: (1) Community environments, (2) Research infrastructures, and (3) Philological projects. Within the first group Interedition, Bamboo, TextGrid, and Open Philology are discussed; the second group encompasses DARIAH, CLARIN, and Europeana; finally, the third group briefly introduces projects with a marked philological perspective, id est, the Perseus project, the Van Gogh digital edition, Muruca/Pundit, SAWS, DINAH, NINES, and the Textual Communities platform.

2.2.1 Community environments

Interedition

The Interedition initiative[2], funded by the European Cooperation in the field of Scientific and Technical Research (COST action), aims at establishing a research community for scholarly editing. The project is also supported by the European Society for Textual Scholarship (ESTS) and it is devoted to promote both the interoperability of digital editing tools and a shared methodology in the field of textual scholarship. Interedition fosters the development and availability of web services to make the interoperability among various tools and scholarly resources possible. The solution proposed by Interedition for textual heritage management is twofold: (a) the creation of a sustainable digital edition model, and (b) the development of a sustainable technical framework and infrastructure. To reach these objectives the initiative encourages projects dealing with textual scholarship to develop tools as open as possible and it promotes communication amongst scholars by using innovative and shared frameworks. The Interedition architecture provides a number of cloud web services (called *microservices*), which support specific tasks in scholarly editing. The microservices are designed as a reusable building blocks of

[2]http://www.interedition.eu/

digital scholarly tools. The well-known CollateX (see section 2.3.4), which inherits the Peter Robinson's Collate tool [61], has been developed within this framework. Apart from providing and encouraging the development of tools, several projects involving the creation of digital editions, archives and corpora are under the Interedition umbrella. The majority of them regards individual authors and works, as well as specific historical periods.

Bamboo

Bamboo[3], funded by the Andrew W. Mellon Foundation, is a cyber-infrastructure initiative. The project aims to enhance the research in arts and humanities, thank to shared infrastructures and services. As for Interedition, Bamboo's objective is to assist each step of the scholar workflow, from acquisition and accessing texts, to analyzing and editing them by implementing web services, within a collaborative environment. The challenge of this initiative has been to advance arts and humanities research through the development of shared technology services. Among the technical requirements, flexibility and scalability are the fundamental architecture features to be addressed. Unfortunately, as outlined in [62] the infrastructure development has experienced various challenges: "changing scholars' traditional practices is not effortless". Nevertheless, even if the project was not able to realize the main objectives, it established an important channel of communication among scholars, librarians, computer scientists, and information engineers [63].

TextGrid

TextGrid[4] is a Java-based Web environment, funded by the German Federal Ministry of Education and Research. TextGrid, as suggested by its name, is based on a grid of computers and represents a virtual research infrastructure for philologists, linguists, musicologists and art specialists. It promotes access to, and exchange of, data in the field of DH. This German Framework provides integrated tools for analyzing texts, and gives computer support for digital editing purposes [64].

The system architecture encompasses several interoperable layers. Additionally, it leverages a long-term repository embedded in a full meshed infrastructure connected to Internet. Services and access tools compose the Middleware layer, which, in turn, is used by other modules and software components positioned at a

[3]http://www.projectbamboo.org/
[4]https://www.textgrid.de/

higher level of the system architecture. These latter exploit the Middleware layer to communicate both with the Grid and with the user interface (GUI). The system provides many tools and services. Among others: Editor XML, Text-Image-Link Editor, Dictionary Search Tool, Lemmatizer, Tokenizer, Semantic Search Tool, Metadata Editor, Upload Tool, Streaming Editor, Sort Tool, Classical Philology Gloss Editor, LEXUS and COSMAS, Collationer and Dictionary Link Editor. The development technology platform uses Java programming language, engineering principles for software development, as well as text processing tools, text mining and natural language processing.

Open Philology Project

The Open Philology Project[5] is an initiative within the Humboldt Chair of Digital Humanities at the University of Leipzig. The objective of this project is to enable Greco-Roman culture "to realize the fullest possible role in intellectual life" by means of the rise of digital technology opportunities. The initiative covers three complementary functions: 1. produce open philological data, 2. educate a wide audience about historical languages, and 3. integrate open philological data from many sources. The project focuses on the philological and linguistic perspectives of sources. In this sense, the Open Philology attempts to convert manuscripts and printed works into digital resources as much as possible. Furthermore, it provides tools for annotating, comparing, connecting, interpreting, proving or rejecting hypotheses, finding evidence, studying critical apparatuses and commentaries. Open Philology Project builds upon and supports Perseus Digital Library. Indeed, it contributes to expand open collections and services, while reaching an increasingly global audience. The production of data includes four modules: 1. Workflow module, enabling a digital representation of a written source enriched by linguistic and named entities annotation; 2. Distributed review module, assessing and representing the reliability of data; 3. Repository module, preserving data based on Collections, Indexes, Texts, Extensions architecture (CITE)/Canonical Text Services (CTS) architecture (see section2.4.2); 4. e-Portfolio Module, aggregating and distributing users' contributions.

[5]http://www.dh.uni-leipzig.de/wo/open-philology-project/

2.2.2 Research infrastructures

Digital Research Infrastructure for the Arts and Humanities

The Digital Research Infrastructure for the Arts and Humanities (DARIAH) project[6] aims to develop a European infrastructure and a data platform able to provide and to integrate services for digital arts and humanities [65]. The DARIAH data backbone attempts to be as open and decentralized as possible. This configuration is functional to the creation of federation mechanisms that are efficiently applicable in research-oriented applications across diverse data sources. The principal efforts focus on data analysis, data visualization, and task management. In addition, the infrastructure allows to add any suitable source while granting the possibility to select resources for specific application environment. It results that different sets of data sources can coexist. As far as new application development is concerned, it requires more effort than simply querying an existing database, as these applications operate on the federated DARIAH backbone. Thanks to this infrastructure, scholars have a digital mean to enhance data quality and to obtain long-term preservation and interoperability, as well as to foster data sharing. In order to manage the aforementioned capabilities, DARIAH has established four programs: (1) e-infrastructure; (2) research and education; (3) scholarly management; (4) advocacy, impact, and outreach.

Common Language Resources and Technology Infrastructure

Common Language Resources and Technology Infrastructure (CLARIN) is an international project[7] which aims at creating a stable and extensible research infrastructure for language resources and technologies. In particular, the tools and resources address the humanities and the social sciences research communities [66]. CLARIN defines important technical aspects such as integration, interoperability, stability, persistency, accessibility, and extendibility. Besides, it also specifies a set of procedures that need to be adopted in order to make the above aspects available. CLARIN attempts to implement an infrastructure able to assist researchers in digitalizing resources, collecting and annotating large corpora, dictionaries or language descriptions. This project endeavors to provide a wide, safe and easy access to language resources. The CLARIN backbone leverages secure grid technologies which

[6]https://www.dariah.eu/
[7]http://clarin.eu/

allow the federation of institutions. In this context, the access policy uses Single Sign On (SSO) to log once in the federation [67]. In addition, CLARIN manages a multitude of International Organization for Standardization (ISO) standards describing Language Resources and tools for the sake of interoperability (an overview of the data model is illustrated in section 2.4.2).

Europeana

Europeana[8], co-funded by the European Union, is a platform that enables information and knowledge exchange in the domain of Digital Humanities. The framework serves as a point of access to the digitized cultural heritage objects. These resources belong to different cultural fields and are provided by various institutions in Europe [68]. Therefore Europeana is the result of many funded European digitization projects which can be accessed through it. This project aims to make rich data and functionality available to textual scholars by means of both Application Programming Interfaces and Linked Open Data paradigm (see section 2.6.4 and 2.4.2). Thus, the Europeana portal can be seen as one of the outcomes implemented thanks to the programmatic access to its resources. Europeana attempts to achieve two main objectives: 1. facilitate connection among related data and make them easily accessible through common Web technologies and 2. enable everyone to access, reuse, enrich and share data [69, 70].

2.2.3 Philological projects

Perseus project

The Perseus project[9], hosted at Tufts University, is one of the leading initiative in providing scholars with a suitable cyber-infrastructure. In oder to build this environment, the infrastructure includes two tools: Philologist/Perseids - powered by Son of Suda (SoSol) - and Alpheios. These Web applications allow scholars to perform the version-controlled editing of texts and their linguistic analyses. The Perseus project is the largest archive concerning the Graeco-Roman world and the classical Greek and Latin texts. It provides data and tools for linguistic analysis, such as Treebanks, and annotated entities in Open Annotation Collaboration (OAC) compliant with Resource Description Framework (RDF) format. The

[8]http://www.europeana.eu/
[9]http://www.perseus.tufts.edu/

digital archive provides, among other, also lexical databases. The Perseus archive aims to provide a systematic repository access to every authoritative Greek and Latin author, encoded in The eXtensible Markup Language (XML), in particular following the guideline of the Text Encoding Initiative (TEI) (see section 2.4.1). To do so, this digital library uses the CTS/FRBR data model (see section 2.4.2) to represent different primary sources, scholarly editions and translations of particular works.

Sharing Ancient Wisdom

Sharing Ancient Wisdom (SAWS) is a project[10] funded by the European Research Area (HERA). It aims at establishing a research workflow of editing and publishing TEI/XML-based digital editions of ancient manuscripts [71]. As an initiative, SAWS project is similar to the works illustrated in chapter 4. It includes a linking software module and a semantic enhancement analysis of its target resources. Moreover, the SAWS methodology encompasses TEI markup, CTS Uniform Resource Name (URN) identifier, Semantic relationships among named entities and parallel analysis for Greek and Arabic literary texts.

Muruca/Pundit

Muruca/Pundit[11] is one of the most used and famous integrated platform for studying, annotating, searching, and browsing digital literary materials. Several worthy projects leverage this framework, among others NietzscheSource[38][12], WittgensteinSource[13], and the recent BurckhardtSource[14], Daphnet[15], and the Digitised Manuscripts to Europeana (DM2E)[16]. This environment is extremely customizable while assisting scholars with knowledge creation and knowledge reuse [72]. Muruca/Pundit leverages Open Annotation Data Model (OA) specification and Europeana Data Model (EDM) definition (see section 2.4.2). The system exploits the Semantic Web (SW)technologies like RDF and the HyperText Transfer Protocol (HTTP) Application Programming Interfaces (APIs) for linking and

[10]http://www.ancientwisdoms.ac.uk/
[11]http://www.muruca.org/ - http://thepund.it/
[12]http://www.nietzschesource.org/
[13]http://www.wittgensteinsource.org/
[14]http://www.burckhardtsource.org/
[15]http://www.daphnet.org/
[16]http://dm2e.eu/

consuming data. The critical module in the platform is an annotation server that manages the persistence and the users.

Van Gogh Letters

Van Gogh letters[17] is a collaborative project between the Van Gogh Museum and the Hugens Institute of the Royal Netherlands Academy of Arts and Sciences [73]. The initiative consists of a web platform which supports an advanced browsing process. Scholars are able to interact with a dynamic edition which involves diplomatic, facsimile, notes and artworks. The sources are marked up by using the TEI scheme (see section 2.4.1).

DINAH

DINAH is a Web-based philological platform, aimed at the construction of multi-structured documents [4]. The system provides scholars with a suitable multi-perspective annotation system. The platform provides a mechanism for managing overlapping hierarchies. This tool allows scholars to annotate multi-structured textual phenomena in a dynamic way. Indeed, this strategy fosters an efficient management of complex documents. Within DINAH platform scholars are free to create new structured vocabulary. This means that the available markup elements are not fixed and static.

NINES

Networked Infrastructure for Nineteenth-century Electronic Scholarship (Nines)[18] is a scholarly project whose objective is to create a coordinated network of peer-reviewed content in order to obtain a comprehensive framework for scholars. The project is developing a number of valuable tools aimed at processing electronic resources and at promoting digital initiatives for textual scholarship [74]. Among these the Collex tool and the JuxtaCommons (see section 2.3.3) are worth mentioning as they assist scholars in working on the interpretive studies of texts. Finally the platform allow users to teach and research in a online environment.

[17]http://www.vangoghletters.org/
[18]http://www.nines.org/

Textual Communites

Textual Communities [19] is a collaborative editing environment based on a social media software. It has been designed to handle different cultural perspectives of the studied materials namely the text, the document and the work perspective. This project was born within the Interedition initiative (see section 2.2.1). One of the critical aspects that the Textual Communities system handles is the double representation of digital sources as text marked upon a document and as text structured into communicative acts. In this way, the formal scheme, called *DET*, developed for describing documents, texts and works identifies every part of each textual entity, in every document, text and work. It also defines how textual elements relate to each other. It is worth noting that both the document and the text are completely hierarchical for the system model. this means that each successive object within a sequence of textual objects must be contained within the preceding object. For example a line is part of a page, which is part of a document. The Textual Communities environment leverages for the persistence the document-oriented MongoDB system and JSON objects for data exchange and serialization purposes. The system provides functionality for comparing different transcription revisions and the collation of the text in all the documents.

2.3 Textual scholarship tools

Word processors have been introduced as the first computer programs for textual editing: they can be easily used to produce complex formatted documents. However, they are not suitable to deal with specific and complex activities such as the construction of critical editions. Several applications and tools have been developed over the years to handle digital documents within a philological perspective or for publishing and delivering purposes [75]. The outcomes have achieved great results and benefits for both classical and computational scholars. It follows a list of the most renowned tools and applications. It concerns excellent products and admirable experimentations on editorial critical problems and sophisticated solutions envisaged to obtain electronic editions. Nevertheless, the relationship between the effort that is needed during the encoding operations and the consequent results has not been considered so beneficial by all communities of scholars, at the time of writing [3].

[19] http://www.textualcommunities.usask.ca/

2.3.1 TUSTEP - Tübingen System of Text Processing

The Tübingen System of Text Processing (TUSTEP) is one of the most used and discussed environments for textual scholarship[20]. It has a long history, which dates back to the late '60s, when Wilhelm Ott at the University of Tübingen conceived this tool [32]. TUSTEP exploits Fortran and C programming languages and it is characterized by a complex framework[21]. It operates on a modular basis and can assist different phases of textual scholarship workflows. This tool supports scholars in making critical editions, starting from a semi-automatic collation phase. In the end, TUSTEP transforms the collation outcomes into a critical apparatus and publishes the edition both in printed and electronic versions. As mentioned, the collection algorithm does not provide a complete automatic procedure. In fact, users need to run a separate application in order to fix the more problematic readings. Recently, a new tool, Tübingen XML-based scripting language for scholarly text data processing (TXSTEP), has been launched. It is an XML-based front-end built on top of the TUSTEP architecture. The main objective of TXSTEP is to provide an "up-to-date established syntax using a more friendly interface in terms of XML editing" [76]. TXSTEP is a sort of XML-based API towards the classical functionality provided by the TUSTEP framework. To conclude, the TUSTEP /TXSTEP environment offers useful services, like indexing features and pattern matching functionality. Nevertheless, the system presents several restraints, such as the heavy learning curve and the difficult to use.

2.3.2 LaTeX / Mauro-TeX

Maurotex[22] is a textual scholarship tool developed from the well-known latex typesetting system. It can be described as a markup environment devoted to cope with the whole philological process [77] (i.e. starting from the transcription of the witnesses and ending with the publishing of the critical edition). The project aims at publishing the critical edition of Maurolico's works both in a traditional printed and in an electronic version. Maurotex allows scholars to obtain the electronic version of the work through an HyperText Markup Language (HTML) or a Portable Document Format (PDF) output. In addition to providing the Latex style mark-up, Maurotex also disposes of a more suitable XML encoding style

[20]http://www.tustep.uni-tuebingen.de/tustep_eng.html/
[21]TUSTEP requires training in order to be used, and relies on outdated standards.
[22]http://www.maurolico.unipi.it/mtex/mtex.htm/

[78]. Concerning the Latex Style, the basic idea of MauroTex is to identify variant readings thanks to a specific macro function (\VV{}). This macro can convey an indefinite number of fields nested into the main brackets. Each field represents a variant reading and can be used to build the *stemma codicum*. In addition, further shorthands are available in order to handle the critical apparatus with the philological standard arguments, such as interlinear additions (\INTERL). Finally, language provides a set of macros that include basic textual-phenomena, such as omissions, conjectures, integration, expunctions, trivial variant readings, pages, titles, and so on. The macros functions rely on other latex packages specifically developed for critical edition such as ledmac/edmac, ednotes or bigfoot[23].

2.3.3 JuxtaCommons

JUXTACommons[24] is an open-source and web-based tool aimed at handling collation and producing critical apparatus. At the time of writing, it is actively developed by using Java programming language and it seems the most advanced framework among similar initiatives. JUXTA environment, developed within the NINES initiative (see section 2.2.3) by the University of Virginia, allows users to upload different types of inputs. It provides capabilities for collating and visualizing digital sources by means of browsers. In addition, the tool provides several user-friendly interactions and functionalities. The collation phase relies on an adaptation of Heckelâs algorithm [79] as well as on additional tools which have been developed within the Interedition project (see section 2.2.1). The system produces XML-TEI collation within the parallel segmentation modality. Several visualization diagrams such as histograms and maps are provided. It is worth noting that the framework exposes Web services API to exchange data and to share functionality.

2.3.4 CollateX

CollateX[25] is a philological tool developed in Java programming language [80] which aims to effectively work with textual variant readings. In addition, CollateX is an ongoing environment which actively collaborates with the Juxtacommons

[23]For further detail please visit https://ctan.org/pkg/
[24]http://juxtacommons.org/
[25]http://collatex.net/

project (see section 2.3.3) within the Interedition initiative (see section 2.2.1). Consequently, it offers a valuable framework to perform collation work. The tool allows users to change collation algorithms implementing at least three interchangeable collation solutions that use progressive alignment techniques (see section2.6.2):

- Dekker algorithm [80], which aligns an arbitrary number of texts detecting transposition and local alignment;
- Needleman-Wunsh [81], which represents an optimal algorithm for global alignment;
- MEDITE, which is an experimental implementation of the Bourdalliet and Ganascia algorithm [82].

The tool outcomes can be obtained in various data formats, such as JavaScript Object Notation (JSON), XML-TEI, GraphML, GraphViz, and Scalable Vector Graphics (SVG). In addition, multiple web services have been developed under the umbrella of Interedition (see section 2.2.1). A noteworthy aspect is that the data structure used by the application is the “variant graph” (see section 2.4.1). The design of CollateX has envisaged a core library, which gives the opportunity to embed the package into external applications.

2.3.5 Text::TEI::Collate

Text::TEI::Collate[26] is a promising, open-source tool, which the University of Oxford is developing for scholarly text collation. The program, implemented by using the Perl programming language, is highly modular and has a well-designed internal structure. The outcomes of the process can be provided in several formats as strings, JSON, or XML-TEI in parallel segmentation format. The design of this tool promotes the implementation of useful APIs. Indeed, the Text::TEI::Collate seems to be conceived as a coherent library for textual collation. Since it has been developed at the moment of writing, the algorithm is still basic and does not handle transposition.

[26]http://search.cpan.org/ aurum/Text-TEI-Collate-2.1/lib/Text/TEI/Collate.pm/

2.3.6 eXist-db

eXist-db[27] is a native XML Data Base (DB) system implemented in Java and freely available in open source [83]. The system is modular and largely used within the community of digital humanities since its fundamental unit of storage are XML documents. As a matter of fact, it increases performance for expensive eXtensible Stylesheet Language Transformations (XSLT) transformations and XPath queries, which are typically applied to documents encoded by digital scholars (see section 2.4.1). eXist creates a Document Object Model (DOM) tree from the XML document and provides index mechanisms with regards to the structure of elements and attributes, string search and full text search (see section 2.5.1). In addition, the indexing scheme identifies relationships among nodes in the document (elements, attributes, text, and comments). The framework provides a RESTful end-point, through which developers and researchers can easily access and modify remote documents.

2.4 Data and metadata encodings

Many technologies exist for representing and processing digital documents and texts. In order to develop a computational environment for textual scholarship, it is fundamental to evaluate at least three essential points:

- standardization of data and metadata formats for literary and philological studies;
- modelling of domain ontologies for semantic description of the cultural objects;
- services for the data exchange among working groups using common protocols and identification of textual units.

These three elements collaborate to the interpretation and description of different features contained in textual materials. Digital texts and documents are the result of different levels of encodings, ranging from characters up to semantics and pragmatic.

[27]http://exist-db.org/

In the first place, this section will touch upon Unique, Universal, and Uniform character enCoding (UNICODE) standard. It will then discuss the hierarchical model theorized by Renear, Mylonas and Durand [84].

While several markup languages for digital documents exist (PDF and HTML are examples of general-purpose formats for encoding texts), the most common language adopted for encoding cultural heritage documents is the XML. Generally, the XML vocabulary suitable for literary computing makes use of the guidelines drawn up by the Text Encoding Initiative (see section 2.4.1). However, the hierarchical structure of cultural heritage documents is faced with issues, as for instance overlapping and multi-version [85]. The variant graph attempts to solve similar matters [86, 87]. On the one hand, the identification of textual units is formalized by the CITE/CTS architecture. This architecture associates a URN to every textual entity of any specific edition. On the other hand, the conceptual model developed by the Functional Requirements for Bibliographic Records (FRBR) allows modelers to move beyond printed book versions, by tracking the logical units within and across the traditional works. Various institutions have proposed models and solutions, concerning ontology and domain conceptualization. For example, Europeana and CLARIN have formalized their data model, as witnessed by the EDM and the CLARIN Metadata Infrastructure (CMDI), respectively. In conclusion, Web-services facilitate data exchange among working groups distributed world-wide, using common protocols such as OAI-PMH, OAI-ORE. With this rationale, a survey of data and metadata approaches is proposed below. This covers standards of description which are directly influenced by the nature of textual objects.

2.4.1 Data

UNICODE

Text refers to a stream of characters [88] which are represented according to some numeric mapping. Consequently, single binary code represents a specific character. This latter, in turn, can be identified with a related symbol.

Characters, within written resources, are symbols pertaining letters, punctuation, or diacritic marks. In this framework, two levels of information have to be considered: a) abstract symbol b) specific shapes (glyph). As a result, this policy is mainly designed for information exchange, rather than for appearance purposes.

Nowadays, since classical and historical texts cannot be well-encoded by the ASCII standards, cultural texts have to use UNICODE character set as it covers nearly all writing systems [89]. The most adopted UNICODE encoding versions are the fixed-width two-byte encoding (UCS-2) and the variable-length one (UTF-8).

UNICODE, which joined the Universal Character Set (UCS) standard (ISO/IEC 10646) [90], guarantees an easy adding of new scripts as well as language independence. It defines a *code-space* of 1.114.112 potential code points divided into 17 planes, each plane consists of 65.536 code points. For example, the Basic Multilingual Plane (BMP) is the *Plane 0* ranging from $U + 0000$ to $U + FFFF$ (Code points in the BMP use just four hexadecimal digits $U + nnnn$). The BMP is an important Plane as it supports the inclusion of old standard character sets (for example basic American Standard Code for Information Interchange (ASCII) codes corresponds to UNICODE code points).

It also provides characters for most of the literary texts. It derives that ranges of code points have been reserved for ancient writing systems, "already known or still to be discovered" [91]. UNICODE code points are handled by using ad hoc encoding maps (Unicode Transformation Format (UTF)). Different kinds of UTF exist, some of them use bit configurations of fixed length and others use variable-length. For instance, UTF-8 consists of sequences ranging from one byte to six bytes; instead, UTF-32 or UTF-4 consists of fixed-length 4-byte sequences. It is worth noting that while UTF-8 is platform-independent, UTF-16 and UTF-32 are platform-dependent format as regards the byte ordering. These are also incompatible with ASCII table. For these reasons, 8-bit encodings are should be preferred for representing text, even if the platform is based on different formats. Moreover, UTF-8 adopts the segment-based management. This means that the binary code for frequent character representation uses few bits, while dedicated bits denote that the character representation is longer. Noteworthy, UTF-8 encodings allow performing string matching in a text file using a byte-wise comparison (which is a valuable help and it allows faster algorithms). UTF-8 represents ASCII values (0-127) using a single byte with the leftmost bit at 0, which entirely corresponds to its counterpart representation. To conclude, typical software for UNICODE text processing, especially in Java, first calls a function to translate a UTF-8 byte stream into a buffer of 16-bit characters, upon which all remaining text processing is done.

Ordered Hierarchy of Content Objects

As stated by [84], the text is an Ordered Hierarchy of Content Objects (OHCO).

The text is ordered because it encompasses linear sequences. At the same time, objects within a text are hierarchical because entities like paragraphs, sentences, and words exist inside one another. This textual model structures the documents in a hierarchical form and it allows a fast processing by computer due to the inherent tree representation. However such a representation manifests some limits concerning single hierarchy perspective. Consequently, the authors of OHCO extended the basic model by introducing two enhancements: OHCO-2, and OHCO-3. According to the latter texts present several separate perspectives (analytical perspectives), each of which is organized hierarchically. In this case, perspectives can contain overlapping structures (for example, page divisions can overlap sentence divisions). OHCO is a controversial model, which generates theoretical discussions on the interpretation and the digital representation of a document [92, 50].

For instance, in [42] it is argued that an OHCO structure "is not a model of the text, but a possible model for its expression". So, the same document conforms to several overlapping structures, each of which is strictly hierarchical. Nevertheless, textual structures are usually more complex than a single hierarchy. In fact, hierarchies are suitable only for certain kinds of texts and documents, for example in printed works, but they do not fit well in texts like notes (see chapter 3 and 4). Moreover, forcing to identify hierarchy structures while encoding historical texts can lead to erroneous outcomes. For example, the representation of variant readings and transpositions is a textual phenomena that overlaps all fixed hierarchies. The variant graph [87] (see section 2.4.1) is an attempt to solve this problem.

Text Encoding Initiative

The digital representation of documents concerns the encoding of various textual phenomena. For instance, the text encompasses sentence divisions or lines numberings. However, common markup techniques that handle information and visual appeal of data do not meet all scholars' needs. In particular, these requirements consist of disposing tools that highlight and analyze textual-phenomena useful to scholar work [93].

Against this background, the TEI[28] pursues the standardization of markup schemes for literary and philological studies. The TEI provides XML schemes and guidelines for text, extra-text, and para-text encoding with bibliographical, linguistic and philological meta-information. More than other XML vocabularies, TEI markup schemes meet scholars' need to encode texts that can be reused as a starting point for further inquiries.

The main benefit of XML, and especially of the TEI guidelines [94] , is on its simplicity, flexibility, readability and customizability. Moreover this tool guarantees a formal approach for validating the data that has been marked up. Consequently, XML provides a standard way to define a set of tags (*vocabulary*) for specific purposes. Moreover, the multiplicity of technologies that gravitate around XML allows to transform (XSLT), publish (XSL-FO), query and address (XQuery, XPath, XPointer), formalize (DTD, XSD, etc), render (eXtensible HyperText Markup Language (XHTML), SVG) and process (Xinclude, XForms, XML Events) structured documents.

It is worth noting that digital texts along with complex annotated data have strongly increased in number and quality also thanks to the role of the XML technologies and services. The TEI guidelines have become the standard encoding system for applications within literary computing projects. Moreover, a large community uses these schemes to encode texts and to develop software that allows to operate on the raw and annotated documents.

Texts marked up in XML-TEI look like HTML documents. In fact, the two languages have similar tags, as they are both based on another markup language known as Standard Generalized Markup Language (SGML). The most important difference is that HTML is conceived to annotate Web documents while XML describes structured or semi-structured data [95].

In fact, XML provides mechanisms to define the markup elements and how to use them in a flexible and formal way. These mechanisms are the Document Type Definition (DTD) and the XML Schema Definition (XSD). Especially, they define the elements for encoding the document and the typed data to be present in those elements.

The TEI also provides elements for encoding both data and metadata by means of two blocks: *header* and *text*. The TEI header contains four major sections: (1) file description, (2) encoding description, (3) text profile and (4) revision history

[28]http://www.tei-c.org/

[95].

- The file description contains information about bibliography and the title of the electronic resource. Moreover it records data about the authors of the work, the place and the time of the publication. It also contains the source description which stores the data as regards the sources of the electronic document.

- The encoding description indicates the relationship of the electronic document to its sources such as selection of texts, text normalization, and levels of encoding.

- The text profile contains information about the languages used, the citation scheme (e.g. chapter instead of pages, or sentences instead of line numbers). It also reports information about the text production.

- Finally, the revision history enlists every change made to the electronic text in order to help future editing works.

Furthermore, TEI organizes the text into separate parts: *front matter*, *text body*, and *back matter*. The scheme consist of about four hundred tags. The core features are available for all types of texts while extended tags are available for specific types of works, such as prose, poetry, critical editions, and others. An important advantage of adopting TEI is that it fosters projects to preserve, share and reuse the encoded resources. Thus it ensures interoperability, regards to both texts and software. However, the use of TEI is not sufficient to solve all issues related to coding complex documents, as in multilingual and multi-witnesses materials, non-standard typography or embedded figures. Moreover, overlapping hierarchies present encoding difficulties in being translated into a hierarchical design without breaking the model itself. However, the advantages of XML stand in scholarly analysis, in long-term preservation and in data exchanging to different computing environments.

Variant Graph

Section 2.4.1 has underlined the challenges experienced when handling elements which denote different views of a document, i.e. sentences against lines in a well-formed XML structure. The reason for these issues is that XML represents tree structures, whereas textual phenomena often require graph structures [86]. This matter is better known as the *overlapping hierarchies problem* [96]. Indeed, XML-based vocabularies cannot easily represent overlapping structures by means of tags. In fact, tags can contain only text or other markup elements in a nested and well-formed way [97]. Overlapping hierarchies are common in cultural heritage text structures. Two types of overlapping can be distinguished: 1. overlapping structures due to text/logical or document/material representation such as sentences or lines in a paragraph, and 2. textual differences in multi-version and multi-witness documents (these deal with insertions, deletions, alternatives and transpositions). These limits have generally been ignored by textual scholarship tools. In a famous paper [87], Schmidt and Colomb propose a solution to this problem that has been widely accepted. They argue that direct-graphs may represent both overlap and variation, by replacing the conventional structure of documents (linear or hierarchical). Additionally, they claim that graphs can be represented by two isomorphic forms: 1. as a variant-graph or 2. as a list of key-value pairs. A way to achieve that is through Multi-Version Document (MVD) system [98]. The MVD model handles all the versions of a work, either when they result from a process of correction, or when they are copied from the original text into several variant versions, or when the latter practices are combined. Overall, interventions on a text can be summarized in four atomic operations: insertion, deletion, substitution, and transposition. When reading the list form of the graph, fragments which do not belong to the desired version are automatically ignored. The structure and contents of this graph is automatically generated by means of *nmerge* program [36]. This is able to detected transpositions and to merge the text that is shared throughout its existing versions. The variant-graph provides a valuable solution to the overlapping problem cultural heritage texts. This solution attempts to create a new model of text, which is based on a more general structure of these tangled documents. An MVD can be represented as a directed graph, with one start node and one end-node. It can also be serialized as a list of key-values, each describing a fragment of text and the relative set of versions. This multi-version texts approach scales well, because while the number of versions increases, the number of fragments increases, but their size decreases, and the size of their version-sets increases.

2.4.2 Metadata

Dublin Core

The Dublin Core Metadata Initiative (DCMI)[29] is a standard metadata scheme aimed at mapping catalogue information from repositories to systems that aggregate digital objects. For instance, Europeana (see section 2.2.2) has adopted this scheme.

The objective of the Dublin Core (DC) is to define a small set of elements and rules that can be used for describing digital resources. The DC scheme is simple and concise oriented to describe Web-based documents. Nowadays, the set of DC core elements consists of fifteen items. They are *Title*, *Creator*, *Subject*, *Description*, *Publisher*, *Contributor*, *Date*, *Type*, *Format*, *Identifier*, *Source*, *Language*, *Relation*, *Coverage*, and *Rights*. However, DC has been used with other types of schemes and in applications demanding more complexity. Tensions have arisen between the minimalist view, which emphasizes the need to keep the environment elements as simple as possible, and the structuralist view, which argues for finer semantic distinctions.

These discussions result in defining qualified and unqualified DC elements. Qualifiers can be used to enhance the detail of an element. For example, Date element can use a qualifier to identify the date format following the *ISO 8601* standard for representing date and time. All DC elements are optional and repeatable, as well as they may be presented in any order. The DC description recommends the use of vocabularies for fields (e.g. for the Subject field), but this is not mandatory.

However, special working groups are discussing a number of authoritative lists for certain elements such as the resource Type field. Moreover the DCMI encourages the adoption of domain-specific rules for special domains such as education and government. Because of its simplicity, the DC elements are largely used in DH. Indeed, there are many scholar projects that adopt the DC scheme either for cataloging or for collecting data. Nowadays, the DCMI has gone beyond the maintenance of the DC metadata set, indeed it promotes the widespread adoption of interoperable standards and develops valuable metadata vocabularies for discovery systems.

[29]http://dublincore.org/

Europeana Data Model

EDMs, which has replaced the old Europeana Semantic Elements (ESE) model, is a popular conceptual data model (aka an ontology [99]). It has been developed within the Europeana project (see section 2.2.2) for the interoperability among cultural heritage repositories. The integration and refinement of heterogeneous knowledge-bases leverage the EDM model since it is designed as a model for collecting, connecting and enriching the descriptions provided by Europeana data providers. In addition, EDM integrates all the well-known standards and well-established ontologies such as the RDF, the OAI Object Reuse and Exchange (see section 2.4.2), the Dublin Core (see section 2.4.2), and others. Consequently, the model is fully compatible with the SW paradigm (see section 2.6.4), and it is able to interoperate among objects and their digital representations, as well as among objects and their metadata. Furthermore, EDM allows to build a semantic layer on top of the aggregated objects.

EDM aims at gathering data for services like Europeana and provides a top-level ontology which is devoted to abstract the underlying models. In this way the expressiveness and flexibility of the various standards is kept, getting consistency among different formats. As described in [100] EDM provides five fundamental semantic relationships: 1) classification, 2) part decomposition of anything, 3) similarity, 4) aboutness, and 5) history of an item. In this perspective the EDM data model is a worth initiative for a metadata properties generalization.

In conclusion, the value of this model is that it fosters cultural heritage communities towards an open, linked, and semantically structured environment compliant with the SW paradigm. This approach largely impacts new projects and new data models, specifically in the DH community.

Functional Requirements for Bibliographic Records

FRBR[30] is an entity-relationship conceptual model developed by the International Federation of Library Associations (IFLA)[31]. It allows scholars to define the minimum requirements of a source bibliographical description. FRBR also defines the purposes of the registration and how it is independently structured from cataloging rules.

[30]http://www.ifla.org/frbr-rg/
[31]http://www.ifla.org/

The model is largely adopted because of the potential it holds to analyze and document numerous aspect of a work hierarchy at different levels of granularity [101]. As a matter of fact, it defines four high-level classes:

- the *work* as a distinct intellectual or artistic creation,
- the *expression* as the intellectual or artistic realization of a work,
- the *manifestation* as the physical embodiment of an expression of a work,
- the *item* as a single exemplar of a manifestation.

Other classes are grouped in two other sets:

- the group which contains the person and entity classes;
- the group which contains the concept, the object, the event, the place classes.

In addition, FRBR fosters the connections among typed data as well as the connection between the intellectual responsibility and the resource concepts. It is suitable for the management of multiple-version documents, and therefore it supports the registration of critical editions. Finally, the model provides a fair management of transactions, which generally requires:

- precise identification of the digital objects;
- the parties involved and their role;
- the action to be carried out on the requested objects such as read, listen, see, save, transfer;
- the parameters which limit it, such as place, time, amount, repeatability;
- the action of whoever uses the service in exchange for acquired rights.

However, FRBR represents the type of metadata required by scholars as it handles intellectual work of the text together with the physical object.

CLARIN Meta Data Infrastructure

The CMDI[32], implemented within the CLARIN research infrastructure (see section 2.2.2), aims to provide a shared access to language resources and tools. This

[32]http://www.clarin.eu/content/component-metadata/

makes CMDI a flexible means for describing, searching and locating relevant data across repositories with different metadata policies. Therefore, this model allows scholars to describe language resources by various shared components. In this sense, the metadata profile can be adapted and reused for other similar objects. The infrastructure is articulated on three levels of abstraction: 1) Data format, 2) Data, 3) Service. Furthermore, CMDI organizes its structure exploiting standards for several aspects such as concepts and relations category (*ISOcat* and *RELcat*), which are described by means of Data Category Interchange Format (DCIF) [102]. The registry for components and profiles is formally defined by using the CMDI CMDI Component Specification Language (CCSL) and exported through REST-based API. Finally, such profiles are available also in XML schemes and can be used for validating the metadata instances. These latter are distributed to CMDI service providers by means of shared protocols such as OAI-PMH (see section 2.4.2).

Open Archives protocols

This paragraph briefly illustrates the data interchange management through the standard protocols within the Open Archival Information System (OAIS)[33]. Textual scholarship tools have to take into account the interoperability issue among repositories. This effort includes the implementation of mechanisms and standards for content description and transmission. OAIS is an ISO reference model aimed at providing a framework for digital repository. It supports metadata ingestion, curation, preservation, transformation and harvesting services. Beside OAIS, OAI-PMH and OAI-ORE are protocols that aim to establish standard-based interfaces to download data. Specifically, Open Archive Initiative (OAI)-Protocol for Metadata Harvesting (PMH) deems metadata harvesting while OAI-Object Reuse and Exchange (ORE) concerns exchanging information about digital objects. In particular, a process based on the OAI-PMH protocol can search by using several criteria and filters. In this way, it allows engines to retrieve only contents related to a specific data or metadata format. Whereas OAI-ORE is a specification which defines a model for the identification, the description and the transmission of digital objects (called *Aggregated resources*) [103]. One of the most useful aspects of OAI-ORE is that a digital object is identified by an HTTP Uniform Resource Identifier (URI) and that digital object can be anything. The Aggregation, which

[33]http://www.openarchives.org/

describes aggregated resources, has a hierarchical and abstract structure, so that the protocol needs a mapping mechanisms and serialization formats in order to get concrete object representation.

Canonical Text Services

The CITE Architecture[34] is a specification envisioned to cite "any word in any version of any text" [104]. The architecture was developed by the Center for the Hellenic Studies, within the Homer Multitext project, but has been applied to other projects dealing with digital editions. It refers to Collections, Indexes and Texts and provides a protocol of citing and linking digital texts and digital objects. The core of the architecture is represented by two kinds of URNs: (1) the Canonical Text Service URN and (2) the Collection URN. Digital resources at different granularities are identified thanks to these two URNs. In addition resources can be associated with each other by means of the CITE Index. The latter component organizes data in a RDF manner (i.e. triples) shaping a graph which consists of a subject entity, an object entity and a verb [105] . Moreover, the texts are linked to related Region of Interests (ROIs) upon which they appear. Thus, the CITE Architecture links together the segments of a digital edition and the documents to their facsimiles. [106].

Actually, the CTS are a data model and a specification which defines remote services for identifying sources and for retrieving fragments of texts by canonical reference [107]. Compared to FRBR data model [56], CTS-URNs allows scholars to cite each word in any version of a text. This policy derives from the traditional citation schemes (e.g. the chapter/verse identification). Scholars, thus, can explicitly cite every version and every word in every version of a work.

The CTS protocol defines the structure for the URNs which is used by the following pattern of informative fragments:

```
"urn":"cts":[namespace]:
[textgroup].[work].[edition/translation/version]:
[passage]:[subreference]
```

While the string "urn:cts" defines the protocol, the other fields have the following meanings:

[34]http://www.homermultitext.org/hmt-docs/specifications/cts/specification.html/

- the *namespace* is the identifier for the higher-level domain for collection types;
- the *textgoup* is the "traditional, convenient groupings of texts such as authors or collections";
- *work* means "a distinct intellectual or artistic creation", i.e. a work that is represented in the text;
- *edition/translation/version* is the identifier of a specific edition of the work;
- *passage* represents the hierarchical reference in the selected work;
- *subreference* points to a string of characters. It uniquely identifies the instance of any character in the selected passage.

This example from CTS documentation provides an evidence:

- The text group Homer has the URN `urn:cts:greekLit:tlg0012`;
- the work Iliad has the URN `urn:cts:greekLit:tlg0012.tlg001`;
- the edition of an English translation of the Iliad,
  has the URN `urn:cts:greekLit:tlg0012.tlg001.mth-01`;
- the passage in line 10 of book 1 of the selected edition of the Iliad would be `urn:cts:greekLit:tlg0012.tlg001.mth-01:1.10`;
- the subreference to the first instance of the sequence of characters Achilles is `urn:cts:greekLit:tlg0012.tlg001.mth-01@Achilles[1]`;

Open Annotation Data Model

OA[35] is now a W3C specification which provides scholars to annotate distributed resources. It includes two existing initiatives, namely the OAC model and the Annotation Ontology.

Annotation activities are a cornerstone process within textual scholarship [108, 44, 5, 109] for the fundamental role they play in the scholar works. Indeed, comments and annotations encompass different levels of information at different granularity (see chapter 3). This activity includes orthographical issues, pragmat-

[35]http://www.openannotation.org/spec/core/

ical analysis, exegetical and hermeneutical comments, sub-character and meta-collection granularity, and it ranges from text-based to multimedia-based documents.

The OAC provides an ontology allowing scholars to yield annotation by using standard Linked Open Data (LOD) technologies, such as the RDF triples. Ontology enables to define three core nodes: a target resource (`oac:Target`) a body content (`oac:Body`) and the annotation itself (`oac:Annotation`). The *Annotation* node is identified by a URI and describes all the annotation properties, as the creation date of the annotation. *Body* nodes can be composed of any type of data which relates to another resource. The *Target* node, also identified by a URI, is the resource related to the annotation *Body* node. The *Body* node can contain natural language comments by defining two properties: "cnt:chars" and "cnt:characterEncoding". In conclusion, the model also provides date-time properties (`oac:when`).

2.5 Related open source software libraries

Modern applications for textual scholarship make use of effective, open source libraries provided by various institutions. Open source projects are usually preferred, as they dispose of a wide range of artifacts that are appropriate to support scholars' work. Since well-designed libraries export suitable APIs, computational scholars make also use of them, more or less consciously, for writing software. Consequently, the section below introduces some libraries dealing with text and language processing. These initiatives share some important design-related features with the textual scholarship library described in this thesis. For this reason, they are frequently used in digital scholar projects (see chapter 4). However, as already stated, the philological domain presents a wider class of problems that can only partially faced by means of methods, resources, and tools developed within the computational linguistics field. Therefore, it is necessary to keep in mind that the libraries outlined in this section meet the needs of textual scholars restricted to some processing aspects.

A universe of digital documents convey textual content through different, and often non-standard, file formats. As it is explained in this section, Tika library attempts to face such a "digital babel fish" [110]. After a first phase in which digital content is gathered, indexing is to be handled. With regards to that Lucene functionality support scholars with a suitable search engine [111]. Annotation can be created automatically by means of GATE and UIMA, two of the most widely acknowledged architecture for text analysis and annotations [112]. It is also noteworthy to mention StanfordNLP, OpenNLP and LingPipe. These latter are Natural Language Processing libraries developed in Java and able to process data from large document collections [112].

Despite their contribution to the field of textual scholarship, these libraries are not exhaustive. In addition to the aforementioned tools, there are other open source Java projects that are very useful when building textual scholarship applications. These include, among others, Weka [113], Mahaout [114], and Carrot2 [115], which provide machine learning and data mining capabilities.

2.5.1 Text processing

Tika

Tika[36] is an open-source library developed in Java falling under the Apache Software Foundation Umbrella. It deals with extracting text and digital objects from a variety of digital documents and file formats, as for instance PDF or document (DOC) files. In addition, Tika provides the necessary services for managing digital information content. As illustrated in chapter 4, Tika positively contributes to textual scholarship applications, due to the wide use of various file formats in the DH field. This library provides a special component, the parser module, that allows to manage documents in specific file formats. As discussed in the chapter 3, Textual scholarship needs an abstraction of the document format by means of an API, through which scholars can manage the low-level data representation. Tika provides this kind of abstraction by means of suitable parsers after recognizing the document media type. Hence, this library is able to extract textual content and metadata from the supplied files. The system architecture has influenced to a large extent the design of the Textual Scholarship Library (TSLib). In fact, Tika relies on a *Facade* component as a single access point to all of its components. These are 1) the parser framework, 2) the MIME detection mechanism, and 3) the language detection. Moreover, this tool leverages extensibility by adopting repositories. In this way new parsers can be added to the system, as well as new MIME types and processing mechanisms. Finally, both graphical and application interfaces allow

[36]http://tika.apache.org/

users to interact with this library allowing its inclusion into other applications.

Lucene

Lucene[37] is an open-source library widely used by the community of DH developers for document indexing purposes. It is recognized as one of the most efficient and flexible solutions for full-text searching. Lucene is able to index textual resources regardless of their nature (i.e. database, non-structured file, formatted file, etc). The available services allow scholars to create catalogs, to build archives, and to conduct advanced search operations on digitized texts. This software library is composed of three basic elements:

- the "index" , represented by a structure referring to all the documents;
- the "document", which is an internal structured representation of the textual source;
- the "field", such as an element of the aforementioned document, consisting of a name-content pair. The field manages the token stream to be indexed.

Users have the possibility to perform complex search operations through a complete binary/boolean syntax. This includes wild-cards, searches in range, and fuzzy operations. In addition, Lucene offers efficient storage and ranking functionalities. The content of a text can be decomposed into fields and stored in a *term-document matrix* by means of the library APIs. Columns in the term-document matrix represent all the fields of a content instance, while rows represent all content instances in the index. The internal index structure is defined by the developer who declares the fields in a scheme and identifies those to be indexed. In addition, the developer can set up a ranking function for each indexed field. The library provides services for various statistical measures such as frequency or *tf-idf*.

Once the index is built, Lucene offers functionalities for retrieving content. Users can issue many query types such as phrase queries, wildcard queries, proximity queries, range queries (e.g. date range queries), and field-restricted queries. The index can be updated during the searching and can be directly loaded into a Java container, providing APIs for programmers. Lucene works under the aegis of Solr platform, another Apache Software Foundation project. This library has

[37]http://lucene.apache.org/

been ported to several other programming languages including Ruby, Perl, and Python.

2.5.2 Language processing

OpenNLP

The OpenNLP[38] library encompasses several statistical NLP tools including a sentence boundary detector, a tokenizer, a Part of Speech (POS) tagger, a phrase chunker, a sentence parser, a name finder and a coreference resolver. The tools are based on maximum entropy Machine Learning (ML) models (see section 2.6.5) and are written in Java. The OpenNLP tools can be used individually, or as plugins with other Java frameworks. OpenNLP maintains a suite of tools for doing many common Natural Language Processing (NLP) tasks such as part of speech tagging, parsing, and, named-entity recognition. Two of the most used components of this library are the *Chunker* and the *Named Entity Recognition (NER)*, both of which use a maximum entropy model to recognize patterns. The basic approach in chunking is to exploit the POS tagger annotations in order to identify simple word sequences. OpenNLP NER recognizes names of different types of entities such as people, locations, organizations, dates, and others. In addition to the default types, which have models available for major languages, it is also possible to create models from scratch. This is a valuable functionality as regards historical and poorly served languages (see chapter 4). The OpenNLP is designed to work in pipe among its analysis tools, supplying the output of one tool into the next.

StanfordNLP

The Stanford NLP Group[39] has also developed a set of statistical NLP tools including a POS tagger, a syntactic parser and a named entity recognizer [116]. Like the OpenNLP library, StanfordNLP leverages Java programming language and it is mainly based on statistical algorithms. StanfordNLP and OpenNLP can be distinguished for their approaches: the OpenNLP tools are designed in order to be used as plugins in other frameworks; whereas the Stanford library is designed primarily to be used as autonomous tool. One of the aims pursued by the Stanford NLP Group is to identify areas of specific language processing phenomena

[38] https://opennlp.apache.org/
[39] http://nlp.stanford.edu/

where accuracy of maximum entropy models can be improved. Two of the most used components of StanfordNLP library are the POS tagger and the Syntactical Parser, but also the NER is largely used. Stanford POS tagger produces output in a simple plain text format through the use of Penn Treebank tagset [117]. In addition, the Stanford POS tagger is based on a work carried by [118]. The Stanford parser is a set of alternative parsing algorithms and statistical models, developed with the purpose of comparing and evaluating different techniques. Syntactic parsing also exploits rule-based grammars and grammar-based linguistic theories. Its goal is to capture the complexity of a sentence concerning its linguistic features (heads and dependents, subjects and complements, modifiers, agreement, etc.). Parsing task can be performed with un-lexicalized, probabilistic and context-free grammars or with lexicalized grammars too. Finally, the Stanford named entity recognizer is based on conditional random field models [119].

LingPipe

LingPipe[40] is one of the most used linguistic toolkits aimed at building natural language processing applications. Despite the library is a commercial tool, it is free licensed for research use. The LingPipe API is tailored to abstract over low-level implementation details to enable components such as tokenizers, feature extractors, or classifiers to be used in a plug-and-play manner. The API that it exports includes functions to build classifiers, extractors, and other search applications.

2.5.3 Text architecture

GATE

General Architecture for Text Engineering (GATE)[41] is a Java open-source and modular framework for text mining, named entity recognition, information extraction, parsing, ontologies handling, semantic annotation, sentiment analysis, and evaluation task [120].

GATE provides cutting-edge functionality, which include processing multiple languages and large collections of unstructured text. One of the aims of this toolkit is to provide services that support the user in organizing tasks related to annotation. Additionally, GATE provides ML systems and supports the integration of

[40]http://www.alias-i.com/
[41]https://gate.ac.uk/

various NLP as well as ontologies tools. This library exposes effective APIs for extracting and processing sentences, entities, and tokens from text. The system architecture includes two basic sets of resources: 1) Language resources and 2) processing resources. At first, a Language Resource can be a single text (*Document*) or a collection of texts (*Corpus*). GATE separates structured document from its markup using standoff annotations. Secondly, a Processing resource is a specific processing component, such as a tokenizer or a NERs. As a result a GATE application is a collection of processing resources organized into a suitable pipeline. Each application can be named and saved separately. Annotations can also be created with user-defined pattern-matching rules (with a Perl regular expressions syntax), called *JAPE* rules. This allows rapid development of new components without the need of programming skills. The system also provides a wide range of name lists (gazetteers) for named entity lookup. In addition, it includes a built-in information extraction system called *ANNIE*. GATE is a general-purpose system, thus it is not specifically designed for any genres of texts and languages. Processing components, such as NER or POS tagger can be added to the processing workflow thanks to the software plugin mechanism.

UIMA

Unstructured Information Management (UIMA)[42] is an open-source platform for creating, integrating and deploying solutions to manage unstructured textual information. It provides multi-modal analysis and search components. The Apache Software Foundation is in charge of developing the software in Java; whereas the Organization for the Advancement of Structured Information Standards (OASIS) is responsible for defining its standards and strategy. UIMA architecture is designed to be modular and flexible. Each specific component implements interfaces defined by the framework and provides metadata by means of XML descriptor files. Standards, interoperability and scalability are the main priorities of the UIMA architecture. The overall annotation structure is called Common Analysis Structure (CAS) [121], whereas the text to be annotated is called the Subject of Analysis (SofA). XML Metadata Interchange (XMI) is the standard format adopted by UIMA for interoperability. Annotations in the CAS structure have a standard namespace prefixes (<opennlp:Token>) and a dedicates attribute of the SofA element. This attribure conveys the principal text of the analysis. Compo-

[42]https://uima.apache.org/

nents in the platform can be extended and customized in order to perform textual annotations.

Another important part of the UIMA architecture is the type system, which keeps track of the content and type of annotations, still for the sake of interoperability. In fact, by checking the types of annotations, it is possible to verify that the type of component output is appropriate to input in the next component. A sequence of annotators, put together to accomplish a complex task, is managed by creating an aggregate structure. Finally, annotation tools are available as plugins and they include a tokenizer, a part-of-speech tagger, a regular-expression annotator, and a dictionary annotator. As previously mentioned, these tools can be combined to build complex applications.

2.6 Suitable information technologies

Technologies have informed researches in the area of DH in general, and researches in the field of textual scholarship in particular. They can be summarized as follows:

- Document, text and character encodings;
- Algorithms for string manipulation and for text alignment, and vector space models;
- Image processing;
- Machine Learning approaches for text processing;
- Methods and technologies developed in the Linked Open Data (LOD) and SW fields;
- Software engineering principles and processes;

2.6.1 Document, text and character encodings

Methods and technologies concerning the non trivial and debated issues of encodings has been discussed in section 2.4.1.

Cultural heritage documents need at least three levels of encodings: 1) character, 2) document, 3) text. Firstly, the character level deals with the electronic representation of characters, such as the codes and the binary digits which map the symbols of a script. Depending on the adopted encoding policies, characters have different binary representations and different rendering glyphs. Unicode standard (see section 2.4.1) is the leading specification for such an encoding. Secondly, the structure and the information which a document conveys need to be made explicit, formalized and marked up. This means that both physical and logical document data have to be encoded. On the one hand text is composed of chapters, pages, sentences, etc. and, on the other hand, text conveys information about its material support, different variant readings, etc. In this scenario, XML, in general, and the Text Encoding Initiative guidelines, in particular (see section 2.4.1), provide a valuable solution for document and text encodings [122].

2.6.2 String manipulation, text alignment, and vector space model

A text can be represented as a sequence of characters which, in computer science, is called "string" [123, 124]. Hence, functionalities as comparing, aligning, sorting, matching, storing, retrieving, searching, sub-stringing, linking, getting patterns, and computing frequencies are all activities concerning algorithms operating on strings. Computer science, computational linguistics, computational biology (bioinformatics) and textual scholarship largely benefit from strings manipulation techniques [125, 126, 98]. Textual scholarship, in particular, makes use of string algorithms for processing its primary sources as it deals with unstructured data in the form of text. One of the most typical need in textual scholarship is to find strings inside other strings, namely finding all positions of a pattern in a text (substring).

Efficient algorithms available to perform this procedure [127, 128, 129] include pattern preprocessing, like the Boyer-Moore algorithm; finite automata methods; and approaches like the knuth-morris-Pratt algorithm.

Matching algorithms use suffix trees representation as the main data structures. This was introduced by Manber and Myers [130] and it defines an efficient representation of all suffixes of a text. In addition, suffix trees are used in the longest common substring problem and in the identification of all the overlaps which occur in a given set of strings [88]. The so called suffix arrays are adopted to represent all suffixes of a given string ordered on a lexicographical basis. Suffix arrays can be constructed in linear time.

Matching problems constitute only one of the computational issues that may arise from scholarly text processing. Another challenging aspect is, for instance, the traceability of the mutation events that make an original text evolve into different versions (multiple sequence alignment problem [131, 132]). As pointed out by Desmond Schmidt and Peter Robinson in various papers [98, 87, 86, 61, 6], the comparison among multiple sequences to figure out which parts are related and how the sequences evolved from one another is remarkably similar to the text-critical problem of collation (i.e. multiple sequence alignment problem). As an evidence, the bio-genetic problem is really analogous to the comparison of witnesses of a work to bright to light their mutual relationships [133].

One of the most effective algorithms adopted for the alignment of sequences is the Needlman-Wunsch algorithm [81]. It is a dynamic programming solution for global alignment that has been used to align similar sequences of character at different granularities. In this sense, this algorithm is suitable for character-by-character alignment as well as word-by-word alignment. Two other alignment types are worth mentioning: 1) the *local alignment*, such as the Smith-Waterman algorithm [134]. It tries to identify similar blocks in strings that could be widely different; and 2) the *progressive alignment* [135]. It is used to construct phylogenetic tree as it can record the stemmatic relationship among sequences. The alignment algorithms may be used for collation purposes: comparing a number of witnesses of the same text and discovering their relationship; finally, yielding the differences as a critical apparatus [75, 136, 44, 126, 5]. Thus, the outcomes of the collation process can be used to construct a stemma codicum or to visualize a phylogenetic tree [133, 137, 138]. The representation of multiple versions of a same text is an active area of research. The use of dynamic programming and data structures such as direct graphs, suffix trees, n-grams and tries are still under investigation by main researches [80].

Character sequences alignment is often based on the notion of fuzzy matching [139]. This means that similarity measures between textual chunks are a fundamental tool in the textual scholarship environment. The Vector Space Model (VSM) [140] is one of the most effective mean for studying similarity among texts [141]. The model can be applied to term-document or word-context approach, thus resulting in different classes of applications. Based on what has been discussed, it is possible to consider VSM as an algebraic model that maps the terms of a document into an n-dimensional linear space. VSM have several applications such as search/indexing or some aspects of natural language semantics [142, 143] (along with the distributional hypothesis which states that words occurring in similar contexts tend to have similar meanings). Another famous application which makes use of VSM is text clustering. This is an unsupervised technique able to group texts according to their similarity. As a result, similar texts remain in the same group (cluster) while dissimilar texts are placed in different groups [144].

In conclusion, some popular geometric measures of vector distance and similarity coefficients, useful in designing the textual scholarship library, are listed below [145, 146].

- Edit distance (Levenshtein distance),
- Euclidean distance,
- Manhattan distance,
- Hellinger,
- Bhattacharya,
- Kullback-Leibler,
- Dice coefficient,
- Jaccard coefficient,
- Jensen-Shannon.

Determining the most appropriate similarity measure dependents on the specific task. Options can range from the sparsity of the statistics, to the frequency distribution of the elements, as well as the smoothing method applied to the matrix. For further details on these measures please refer to the wide literature.

2.6.3 Image processing

Image processing heralds significant applications in DH studies, due to the primary role facsimile reproductions and image enhancement play in the production of modern electronic editions. Indeed, the digital image of documents, concerning mainly valuable manuscripts (see chapter 3), give scholars a mean to gather primary information conveyed by the sources. Actually, the transcription of a document is just a part of the textual scholarship process. Consequently, tools for image processing, for instance related to brightness adjustment, magnification, deskewing, dewarping, despeckle, layout analysis, segmentation, content detection, etc. are effective in clarifying difficult readings. Thus, image processing allows to enhance the legibility of texts even on strong damaged documents [91, 147, 148]. Dated initiative, such as the Beowulf project [149, 150], as well as more recent ones, such as the Cultural Heritage Imaging Organization (CHI) [151], have achieved valuable outcomes in image enhancement. One of them, for instance, involves the ability of scholars to detect how many copyists wrote a manuscript, thanks to pattern-recognition techniques [152]. In this manner, digital images help scholars to improve the readability of the textual passages and reconstruct the controversial readings by showing clearer words.

Image processing for documents encompasses mathematical functions which implement suitable convolution filters. However, this topic has been given limited attention in the context of this study (see chapter 4). Further information with regards to image processing can be found in the supplied literature.

2.6.4 Linked Open Data methods and technologies

Cultural Heritage documents are objects carrying meaningful concepts. These concepts can be used to search and browse as well as to foster further inquiries in the field of DH. As a consequence, LOD principles are largely adopted within textual scholarship applications. Linked Open Data, especially when they are open and semantically characterized, have gained importance for literary computing. Indeed, recent technologies offer a new level of flexibility, interoperability, and relationships for digital repositories [153]. Generally, relational database systems prove too rigid for integrating data sets in the field of humanities, which is often heterogeneous and redundant. Consequently, the collection of documents with annotations, comments and metadata have to leverage data formats able to easily be integrated in other frameworks.. For instance, RDF is the standard technology for implementing shared and linked data on the top of the open web platform. By doing this, it is possible to better integrate heterogeneous collections, cross-linguistic repositories, and multi-cultural archives [45]. In addition, LOD methods enable shared vocabularies and annotation mechanisms, overcoming interoperability issues. The starting step concerns the publishing of metadata as linked data. This consists of encoding information by using RDF format together with resource URIs. Further, linked data sets need one triple store to be freely exposed.

LOD applications come together with two other elements with regards to initiatives in theDH: a) Ontology definition and b)NER / Term Extraction (TE).

The first one is in charge of formally structuring the knowledge present in a specific domain of interest. The second one is in charge of automatically extracting and recognizing, on the one hand, the named entity, such as places, dates, people, organizations, etc., on the other hand, the terminology of the domain. The NER is a NLP task which provides textual scholars with useful applications. For example, a recent Italian research project about the first world war [154] well illustrates such technology. In addition, by combining NER with the SW methodologies, DH researchers find great advantages and enhancements in this field [58, 155]. Basically, entities within the LOD framework have to follow four principles [156]:

- Using URIs as names for things;
- Using HTTP URIs for the sake of accessibility;
- Using standard technologies such as RDF or SPARQL for encoding information;
- providing URIs along with others URIs.

Several initiatives provide entity repositories and APIs available within the LOD cloud, which export semantically characterized descriptions. Among the others, it is worth mentioning *DBpedia* and *VIAF*. These APIs take a text fragment as input and then link the extracted entities to the LOD cloud [157]. For instance, *DBpedia Spotlight* tool is able to annotate entries of DBpedia and to link unstructured information sources to the LOD cloud by means of DBpedia URIs.

The LOD framework is actually an open research topic in the context of cultural textual studies. Indeed, there are several conceptual discussions concerning it. For example, the matter addressing whether a URI identifies an entity or it identifies a document that talks about that entity is still misunderstood. This discussion is critical for DH applications, as it is briefly discussed for CTS notation in the section 2.4.2. Actually, different descriptions of the same reality can lead to ambiguous outcomes, though mechanisms for entity refining and reconciliation have been already developed. For instance, it is possible to handle cross-referencing URIs with the OWL property like `OWL:sameAs`. Despite justified skepticism, LOD technology gives actual benefits to the textual scholarship framework in the SW perspective. In fact, it provides semantically enriched digital object with well-known vocabularies and thesaurus [72].

2.6.5 Machine learning approaches

ML techniques are widely used and studied in computer science in order to solve classification tasks. Several areas of study, which are close to literary computing, such as computational linguistics, text mining and document processing, take advantage of these kind of statistical algorithms. To this regard, in the last decade, experiments and researches adopting ML algorithms have extended rapidly across applications for textual scholarship. The applications range from Optical Character Recognition (OCR) (see section 4.1.1), to lemmatization (see section 4.4) [158] [159], or indexing [160, 161, 162]. As illustrated in section 2.5, several libraries provide API towards efficient solutions for the use of learning strategies.

ML techniques are grouped into two main categories: 1. Supervised approaches, and 2. Unsupervised approaches. The unsupervised ML approaches deal with unlabeled data, such as clustering applications. These kinds of techniques have been briefly introduced in section 2.6.2. The supervised ML systems encompass, in turn, two components: 1. the *learner*, and 2. the *classifier*. The learner builds the classifier (such as the statistical model) from a training set of examples, which establishes both the input of the classifier and the corresponding output. The resulting classifier is then used to assign classes to the input instances, where the values of features are known, but the value of the class is unknown. For example, sentence detection task generally classifies punctuation as two different classes: (a) sentence boundaries, (b) other. Hence, appropriete training sets strive to contain a sufficient quantity of features which can model the domain data, in order to give actual discriminative properties [163]. Therefore, the learning step generally involves a systematic and manual annotation which aims to produce a training set as close to the properties of the input raw data as possible.

Many different methods have been developed as classification algorithms. These include *Naive Bayes*, *Bayesian networks*, *Decision Tree*, *K Nearest Neighbour*, *Maximum Entropy*, *Support Vector Machines (SVMs)*, *Hidden Markov Models (HMMs)*, *Artificial Neural Networks (ANNs)*, just to cite the more famous[43]

These allow to model several applications in literary computing as a classification problem. For instance, ANNs are the most commonly used models to develop Optical Character Recognition engines [91, 152], while Natural Language Processing tasks exploit nearly all the ML techniques.

[43]For further details about Machine Learning techniques suitable for textual scholarship issues please refer to the wide literature

These kinds of techniques presents several advantages, among which the possibility to recognize unknown words. However they also present disadvantages, as in the case of false positives found when words are too similar to the supplied training set.

Among the methods introduced above, HMMs have been investigated (see chapter chap:experiments:annotations) in order to develop a POS tagger for the Latin lemmatizer [164] by using the *HunPoS* [165] tool. In this case, the learning phase has exploited the Latin treebanks [166], which provide manually annotated linguistic data for syntax, morphology and lemmatization.

Essentially, HMMs are probabilistic and discrete dynamic models which derive from the n-order Markov chains [167]. They include two layers of states: (1) a hidden layer and (2) a visible layer. The stochastic process concerns a sequential transition between states evaluated by a set of conditional probabilities. This process describes the sequential shifting called "transition probabilities". The innovation of the HMM model resides in the random output provided by the "emission probabilities" (observable state). In this way the sequence of outputs exhibit information about the sequence of unobservable states. Furthermore, the *Viterbi algorithm* [168] is used to approximate the most likely states sequence (likelihood) that produce the observed input data sequences.

In conclusion, textual scholarship uses machine learning techniques in order to discover new information by detecting patterns and trends. This allows textual scholarship to benefit also from research in text mining, which aims to generate new information holding predictive value [169].

Among the applications for text analysis and text mining that textual scholarship tool can use, there are:

- term disambiguation;
- text classification and clustering;
- event extraction;
- relationship extraction;
- term extraction;
- term normalization;
- text reuse.

2.6.6 Software engineering principles and processes

Software engineering is a discipline of engineering science that studies the nature of software, approaches and methodologies of large-scale software development [170]. Moreover, it addresses theories and laws behind software behavior and software engineering practices [171]. The design of the textual scholarship library is based on practices and principles of software engineering. Computer science and philology, on the one hand, provide textual scholarship with the theoretical foundations, software engineering, on the other hand, focuses on analyzing, designing and developing the software for the TSLib in a controlled and scientific way [172]. Chapter 3 discusses the choices and the results carried out, by taking into account software principles, design techniques and processes.

Chapter 3

Methods

This chapter illustrates the design process and the technological methodologies that underpin this research.

3.1 Introduction

The object-oriented paradigm has been the mainstream approach in software development for the last decade, but its use among the community of textual scholarship has been rather modest [41]. Indeed, few initiatives have been devoted to the requirement analysis of scholarly work as well as to the creation of domain models [173]. As a matter of fact, the object-oriented paradigm facilitates the process of conceptual modeling in two ways: (a) obtaining a better understanding of the specific problems, and (b) keeping the design independent from technologies implementation. Furthermore, conceptual modeling makes use of abstraction, and the object-oriented paradigm provides direct support for promoting modularity and component-based approaches.

Against this scenario, this research presents a number of software engineering techniques and principles that enable the development of textual scholarship tools. The long-term objective is to implement the Textual Scholarship Library (TSLib) by means of well-designed components, open and distributed throughout the community. This significantly improves the reusability of software applications and, consequently, the general outcomes of the researches for textual scholarship [174, 175]. In other words, TSLib components constitute the fundamental building blocks of the general architecture. Unfortunately, the definition of component seems not to be universally accepted. In the context of this work, a component meets the Object Management Group definition:

> A component is a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces [176, 177]

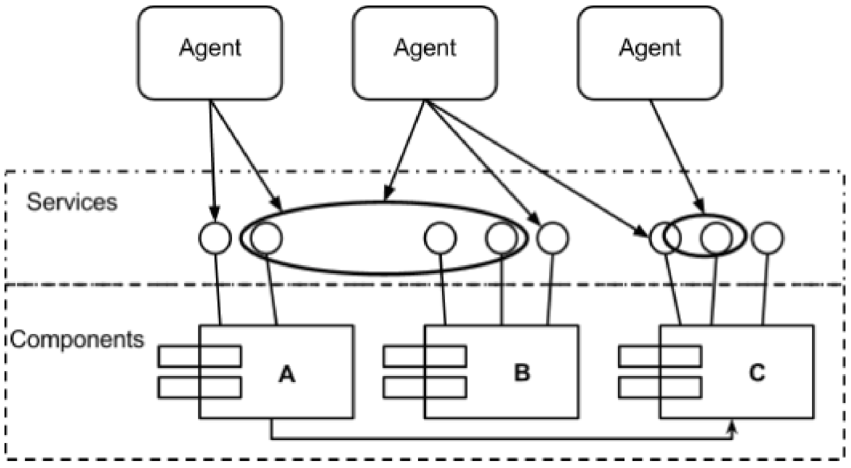


Figure 3.1: Component services example

Each TSLib component is a piece of software devoted to a clear functionality. In turn, each component can be assembled with other pieces of code providing a more complex functionality (Fig. 3.1). Hence, components have the same fundamental principle of the object-oriented paradigm. Encapsulation, modularity, and unique identities are all basic object-oriented principles that are also common to TSLib components. Nonetheless, components in the library are deeply characterized by the possibility to be reused in a number of different contexts in a flexible[1] way [178].

The above mentioned definitions let emerge several important properties [179] for the design of the textual scholarship library. Noticeably, the first property

[1]Two measures characterize flexibility: coupling and cohesion. The first measures the strength of interconnection among software components, that is the degree to which each component depends on the others. The second measures the functions congruity of a single software component. Software should tend to be loosely coupled with high cohesion. This allows components to be used, understood, and maintained independently form each other.

is composability. Indeed, TSLib is constituted by several components which, in turn, can be recurrently divided into sub-components (see chapter 3.3). The second property is reusability. It concerns how components are developed by a component provider. Reusability defines how existing components may be integrated in new applications. Thirdly, modularity and encapsulation properties refer to the component as a collection of services that are related by means of common behavior and data. Finally, the interaction property entails information hiding and access mechanisms to the internal procedures of the modules. In this context, interfaces are the contract for activating the exposed services and for receiving the correct type of outcomes.

Designing software for textual scholarship, which strives to be based on interrelated components, requires a suitable method of modeling as well as a development process [180]. Nowadays, Agile approaches [181, 182] seem to be more effective than the traditional ones (such as waterfall, V-model, Spiral, etc.). Therefore, communities and developers of Digital Humanities (DH) applications, substantially choose Extreme Programming [172] to prototype their algorithms and to release their applications. The method adopted for designing the TSLib, on the contrary, has relied on Agile principles, but also on more classical engineering techniques. This strategy provides a better framework for designing an abstract model and, although simplified and partial, for obtaining a clear view of the target domain.

Theories and processes borrowed from computer science, e.g. Natural Language Processing (NLP) and bio-informatics, have been successfully employed in computational philology (a few examples are morphological, syntactic and semantic tools, parsing algorithms, alignment strategies). On the contrary shared models and conceptualizations, based on a formalization of the philological domain, have not been sufficiently developed [37]. These matters have required a compromise between a top-down and a bottom-up design approach [183].

- The top-down approach implies that the main use-cases with related user scenarios are drawn up; afterwards, it implies that the high-level object behavior is formalized. The designed library appears as a single component at the first step of analysis. Moreover, its architecture defines a high-level organization as a collection of interacting components where their APIs define the expected behavior (deductive process, divide and conquer).

- The bottom-up approach, instead, implies that requirements for specific projects and pre-existing pieces of software lead the design process. In these circumstances, existing software is generalized and refactored iteratively. This means that the whole system is put together from modules that have been developed for similar contexts, through an activity of composition (inductive process, composition).

TSLib design exploits graphical notation provided by the Unified Modeling Language (UML)[2]. Thanks to the UML, software engineers can graphically develop software modules that both humanists and computer scientists can understand. Thus, the software process of the textual scholarship library encompasses all the development phases i.e. requirements, design, and implementation.

The library design process involves a number of key concepts, which can be summarized as follows:

- the open-source as a collaborative and community-driven strategy;
- modular and component-based architecture for the sake of extensibility and reusability;
- the use-case driven approach for the definition of the scholar needs;
- the object-oriented UML modeling of the domain entities for maintaining technology independence;
- suitable design patterns for solving recurring problems;
- the Application Programming Interfaces as a means of (a) communication, (b) functional entry points, and (c) implementation hiding;
- effective and efficient algorithms for document, text and language processing;
- standard schemes and shared resources for data-integration, data-exchange, data-interchange, and long-term preservation;
- Java technology along with the JavaServer Faces Technology (JSF) framework as the main technologies for implementing the library prototype[3].

[2]The UML is a language under the umbrella of the acOMG which is one of the largest consortium of partners from industry and research organizations in computer engineering fields. Since it is a graphical notation and it is not related to a specific method of development, the UML can be used for specifying and visualizing any software artifact throughout the entire software development process.

[3]The library should also be implemented adopting Python programming language

As previously mentioned, the design of the TSLib follows a use-case driven process. Therefore, the Iconix object model [184] and the Agile approaches [185] have been customized and adopted for the aim of this work. The design of TSLib addresses two relevant principles:

- The TSLib must respond to the actual scholar requirements.
- The Agile environment simplifies interdisciplinary teams in terms of communication and consequently, in terms of implementation efficiency.

The outlined process fosters verification and validation throughout the development (testing and refactoring). In addition, this approach provides all the necessary software documents that characterize the TSLib design and evolution. For instance, the UML notation encourages collaborative activities pertaining to the analysis and to the design of the artifacts [174]. In this way, diagrams and documentation foster community interaction, producing improvements and gathering new requirements. The structure and behavior of the library is kept as abstract as possible, so that it can be better refined, extended and implemented during further works. Indeed, platform independent models allow the TSLib to leverage the most appropriate implementation technology available and enable the system to keep up to date with technology evolution [186, 187].

The Agile-ICONIX method applies only a limited number of concepts, as it strictly follows the principle of separation of concerns throughout an entire project. Additionally, it promotes component refactoring and reuse, and follows typical object technology principles. These are the principles of (1) modularity, (2) encapsulation, (3) information hiding, (4) unified functions and data, (5) unique identities, and (6) incremental development. Once started, the software process is responsible for ensuring that the behavior of TSLib is coherent, and that the components provide the desired functionality concerning architectural design and specification.

Therefore, the development of the textual scholarship library begins with the process of gathering information from the target domain. This phase aims at understanding the scholar context and the functionality that the TSLib has to export, along with the input and interaction mechanisms from and to the artifact. In addition, the resources and the constraints of the target domain state the operations to perform for pre-processing and post-processing activities. The UML use-case diagrams suit this initial survey. As an evidence, the use-case diagram outlines important information that represents scholar perspective during the user-system interaction [188, 189]. Fig. 3.2 illustrates an example of how to draw up a use-case diagram. As evident, it specifies high-level interactions between the system and the users (actors). The actor browses a work, compares the text with the related scanned image, and studies relevant comments and secondary sources. Besides use-case diagrams, computer engineers and scholars can communicate better if they draw mock-up pictures. Actually, mock-ups seem to be more friendly for those users with little technical skills. Fig. 3.3 shows the mock-up image derived from the previous use-case examples. Finally, appropriate objects and their constitutive relations (mainly the "depends", "has a", and "is a" relations) model important aspects of the scholar domain. For instance, Fig. 3.4 illustrates a segment of the design related to comments and their properties (versioning, text-selection, etc).

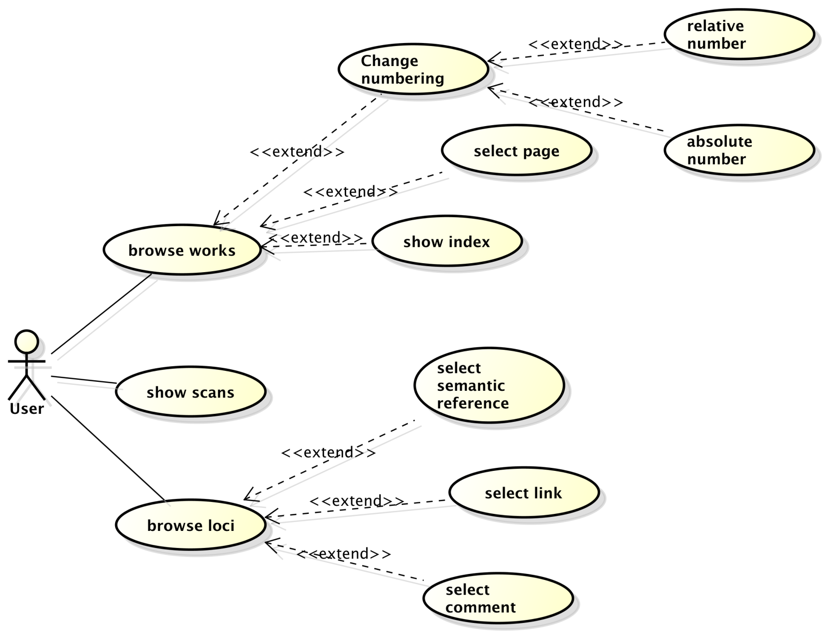
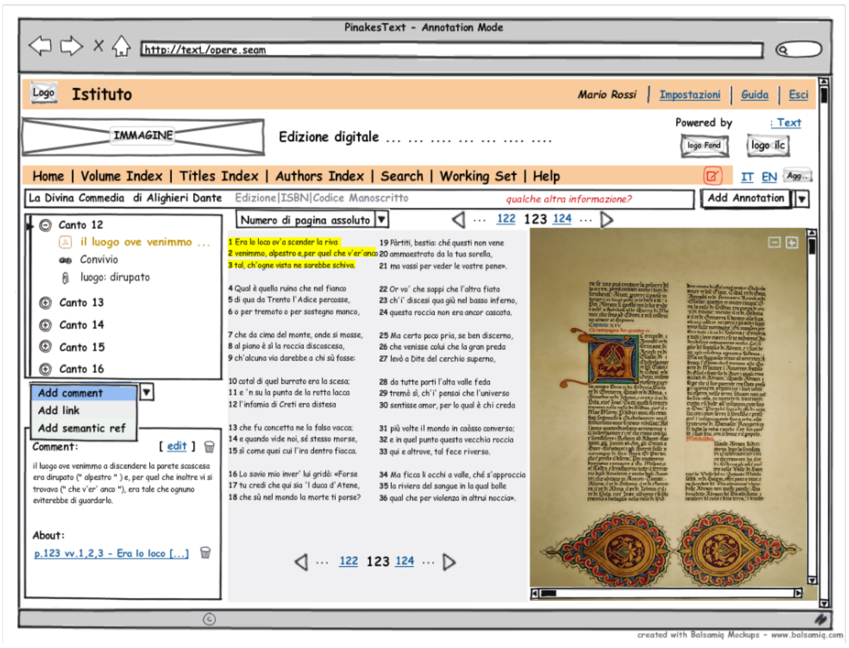


Figure 3.2: Example of UML Use case diagram within the TSLib


PinakesText - Annotation Mode
http://text./opere.seam
Logo
Istituto
Mario Rossi | Impostazioni | Guida | Esci
IMMAGINE
Edizione digitale … … …. … … …. ….
Powered by
: Text
logo Fond
logo ilc
Home | Volume Index | Titles Index | Authors Index | Search | Working Set | Help
IT EN Agg…
La Divina Commedia di Alighieri Dante
Edizione|ISBN|Codice Manoscritto
qualche altra informazione?
Add Annotation
Canto 12
il luogo ove venimmo …
Convivio
luogo: dirupato
Canto 13
Canto 14
Canto 15
Canto 16
Add comment
Add link
Add semantic ref
Numero di pagina assoluto
… 122 123 124 …
created with Balsamiq Mockups - www.balsamiq.com

1 Era lo loco ov'a scender la riva
2 venimmo, alpestro e,per quel che v'er'anco
3 tal, ch'ogne vista ne sarebbe schiva.

4 Qual è quella ruina che nel fianco
5 di qua da Trento l'Adice percosse,
6 o per tremoto o per sostegno manco,

7 che da cima del monte, onde si mosse,
8 al piano è sì la roccia discoscesa,
9 ch'alcuna via darebbe a chi sù fosse:

10 cotal di quel burrato era la scesa;
11 e 'n su la punta de la rotta lacca
12 l'infamia di Creti era distesa

13 che fu concetta ne la falsa vacca;
14 e quando vide noi, sé stesso morse,
15 sì come quei cui l'ira dentro fiacca.

16 Lo savio mio inver' lui gridò: «Forse
17 tu credi che qui sia 'l duca d'Atene,
18 che sù nel mondo la morte ti porse?

19 Pàrtiti, bestia: ché questi non vene
20 ammaestrato da la tua sorella,
21 ma vassi per veder le vostre pene».

22 Or vo' che sappi che l'altra fiata
23 ch'i' discesi qua giù nel basso inferno,
24 questa roccia non era ancor cascata.

25 Ma certo poco pria, se ben discerno,
26 che venisse colui che la gran preda
27 levò a Dite del cerchio superno,

28 da tutte parti l'alta valle feda
29 tremò sì, ch'i' pensai che l'universo
30 sentisse amor, per lo qual è chi creda

31 più volte il mondo in caòsso converso;
32 e in quel punto questa vecchia roccia
33 qui e altrove, tal fece riverso.

34 Ma ficca li occhi a valle, ché s'approccia
35 la riviera del sangue in la qual bolle
36 qual che per violenza in altrui noccia».

Comment: [edit]

il luogo ove venimmo a discendere la parete scoscesa era dirupato (" alpestro ") e, per quel che inoltre vi si trovava (" che v'er' anco "), era tale che ognuno eviterebbe di guardarlo.

About:

p.123 vv.1,2,3 - Era lo loco [...]

Figure 3.3: Mockup example within the TSLib analysis process

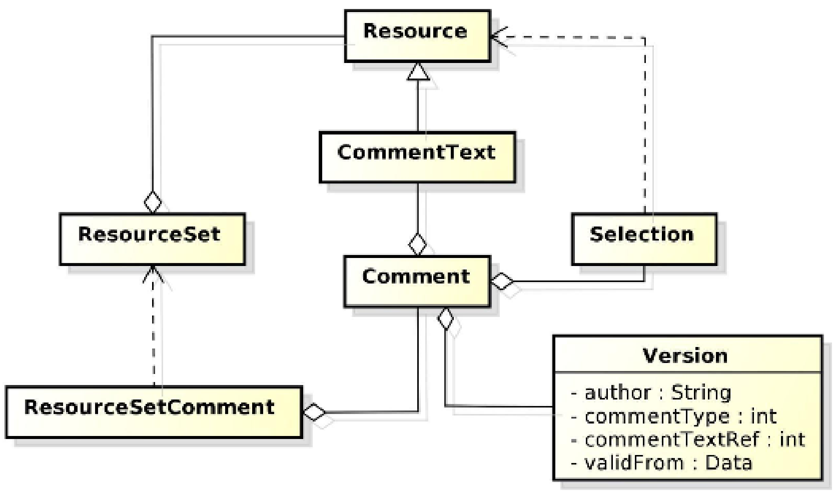


Figure 3.4: Example of UML conceptual class diagram within the TSLib

3.2 Requirements and use cases

The library has the aim of providing all necessary functionalities for the textual scholars, while safeguarding simplicity [190, 191].

As claimed in [51], domain analysis is not a widespread practice within humanities computing. Conversely, in order to understand the services that a textual scholarship library has to provide, it is necessary to perform requirements gathering and use-case definitions. As a main principle, the model should offer a coherent solution to real problems. Therefore, it should be adequately formulated considering the actual needs of the domain. Accordingly, the domain analysis that the library aims to model is the starting point of the TSLib design. This emerges from scholars' requirements and illustrated through a collection of use-cases diagrams.

The definition of the architecture of a complex system for Textual Scholarship should firstly take into consideration the specific features of a textual object, be it ancient, modern or contemporary. Differently from what is commonly maintained, a text is a multilevel, multi-perspective and dynamic entity [136, 44, 50, 37, 42]. Even when appearing in a paper form, for instance, it cannot be considered the final outcome of a stable and finished process. Particularly, as far as ancient works that can be read now in printed editions, the following factors must be taken into consideration:

- variety of copies created at different times, related to the same text (for example, medieval copies executed on handwritten documents);
- presence of variants and/or errors that these copies record, causes often attributed to copyists (ignorance, distraction, randomness, etc.);
- chemical-physical features of the material on which the text is written (paper, parchment, cloth, stone, ceramic, etc.);
- characteristics of the language in which the text was written;
- presence of illustrations and/or pictures that enrich the information present in the text, as in the case of technical or scientific works;
- presence of translations necessary for the comprehension and interpretation of the text;

- possibility of recording annotations, commentaries, or parallel passages needed to correlate the content of a text with similar content in contemporary or previous texts;
- possibility of entering bibliographical information related to the text or to any of its parts.

Although, these features pertain especially to ancient works, they also apply to more recent if not, contemporary works. The latter case is studied specifically by the so called "genetic criticism", which analyzes the formation process that a text, often of cultural value, undergoes before being sent to the press.

Every preliminary version (called "avantesto" by the scholars of this field of literary studies) shows the exceptional mobility and instability of works which have become classics. A demonstration of this phenomenon is present in the archives that keep manuscripts of modern and contemporary authors: many, if not all of these, present many authorial interventions. The corrections, the additions, the erasures are important elements of study for analyzing the stylistical, psychological, and sociological aspects that have influenced the poetics of the author and therefore of his works. These aspects of mobility and variability, which can be called "textual drift" [138] are also attributable to digital texts (following the example taken from molecular biology that has defined "genetic drift" the "random fluctuations in the numbers of gene variants in a population" [133]). These also vary, evolve, change in time thanks to the massive input of the social component of the web. As example of what has been said above, it is appropriate to show a passage taken from F. Nietzsche's notebook containing some personal notes (Fig. 3.5). The features above listed have a direct impact on the system architecture design and on the components to be developed for a model suitable for Textual Scholarship. It is evident that a computational environment, possibly collaborative, capable of assisting a philologist in studying a document of this type must possess at least:

- a browsing module of the digital document;
- a module for transcribing the text contained in the original or facsimile source;
- a module for visualizing notes according to the sequence of the author's interventions, and the cuts that he has performed on this part of his notebook;

- a module for the linguistic analysis of the text (morphological analysis, lemmatization) and the creation of an alphabetical index of the single occurrences, and/or lemmas;
- a module for the bibliographical information, for instance the information concerning the studies made on Nietzsche's text;
- a module that permits the entry of annotations by the scholars that want to intervene on this text;
- a module for the reproduction of the modern edition of the text in printed format, reporting possible notes, translations and indexes.

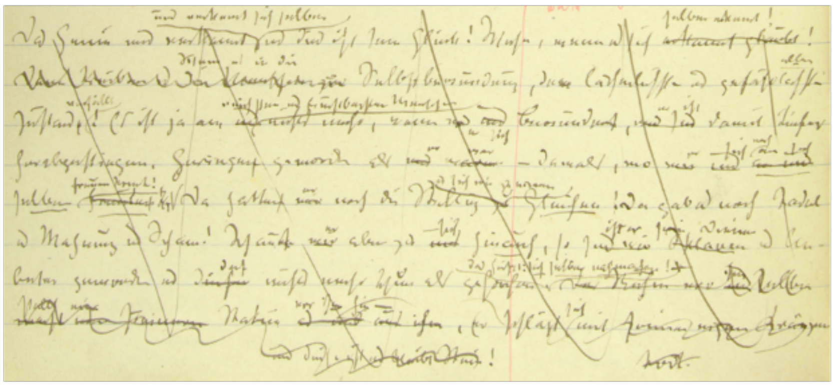

[illegible]

Figure 3.5: Modern manuscript example

This list does not claim to be complete but it is useful for understanding the diverse range of challenges that need to be faced when developing a scientific application for textual scholarship. To date, many research centers have dealt with these issues separately, not considering that the results obtained, although valid, were actually partial. This partial outcome has probably restrained the technological innovation in the field of philological studies. Little application of innovative methods and tools is registered among experts in the field of humanities, mainly due to the computational methods that do not respond exhaustively to the requirements of their studies. As highlighted in chapters 1 and 2 of this study, limited projects for the production of single results (a system for the production of indexes, one for the display of digital images as well as transcriptions and mark-up systems, etc.) have been carried out at present.

On the contrary, this study adopts a holistic point of view of the activities accomplished regarding the scientific (philological) study of a text. Despite the difficulty in reaching comprehensiveness, the TSLib can be incremented according to individual user needs, to the type of philological discipline (epigraphy, papyrology, philology of printed text, etc.) and to the technological components it employs.

The analysis phase tries to clarify the functionalities that the library has to expose. Frequently, in the field of digital scholarship, users require for web applications that simplify document access and offer a collaborative work environment. This scholar environment helps to compare sources, create relations among objects, add notes and comments, edit critical apparatus and share textual materials with the community.

The requirements and the needs of the users for the TSLib derive from the user-stories [192] collected within the Open Philology Project (see chapter 2). Tab. 3.1 shows a fragment of this notable work.

Afterwards, use-cases have been described by means of proper UML diagrams. As a result, use case modeling has identified the core functionalities, relevant actors and roles with regards to the fundamental aspects of the tool. Fig. 3.6 illustrates the principal use-case diagram of the library. It shows four types of actors and the ways in which they interact with the TSLib. Two actors, the **general user** and the **domain expert**, exploit the capabilities of the components, either generally or specifically. On their part, the other users, namely the **developer** and the **domain developer**, can configure and extend the library. In particular the **general user** visualizes and interacts with data, and also performs basic and advanced searches for scholarship inquiries. Correspondingly, the **domain expert** *specializes* the **general user** (i.e. he has the same functionalities as the general user) and can also manage the primary sources. For instance, he is entitled to upload the digitization of the text or perform data acquisition, to process the resources as regards linguistic annotation, proofreading, etc. Furthermore, the **domain expert** can establish both internal and external relations among entities -such as connecting named entities with the Linked Open Data (LOD) cloud (see chapter 2 and 4)- and perform data Create, Read, Update, Delete (CRUD) operations. Finally, the **expert user** handles the information and meta-data regarding the document facsimiles and forms of the document such as stones, rolls, papyri, etc. Fig. 3.6 also shows that the **developer** *specializes* the **general user**. He configures the

1	Manage Content
1.1	As a content manager, I want to add a primary source text to the repository so that it is accessible by users for reading, curation, annotation and research.
1.2	As a content manager, I want to add a derivative work to a repository so that it is accessible by users for reading, curation, annotation and research.
1.3	...
2	Research, Learn, Produce
2.1	As a user I want to view a text and related named entity annotations to explore real word people, places and objects associated with the text.
2.2	...
3	Search
3.1	As a user I would like to find scholia on a work to aid my understanding of the work
3.2	As a user I would like to find references to a place entity within or across works to aid in my study of that place
3.3	...
4	Consume
4.1	As a user I would like to read a primary source text
4.2	As a user I would like to read a translation of a primary source text
4.3	...
5	Curate
5.1	As a user I would like to create a new transcription of a primary source text
5.2	...
6	Analyze
6.1	As a user I would like to get a list of all vocabulary used in a text or set of text and sort it by various criteria (frequency, sequence of occurrence, ngrams, etc)
6.2	...
..	...
...	...

Table 3.1: Example of user-stories extracted from Open Philology analysis work

library by editing the property files, by choosing the engines and adding the extensions. In conclusion, the domain expert developer specializes both the domain expert and the developer user. Therefore, he is in charge of developing extensions to the TSLib offering his contribution as service provider.

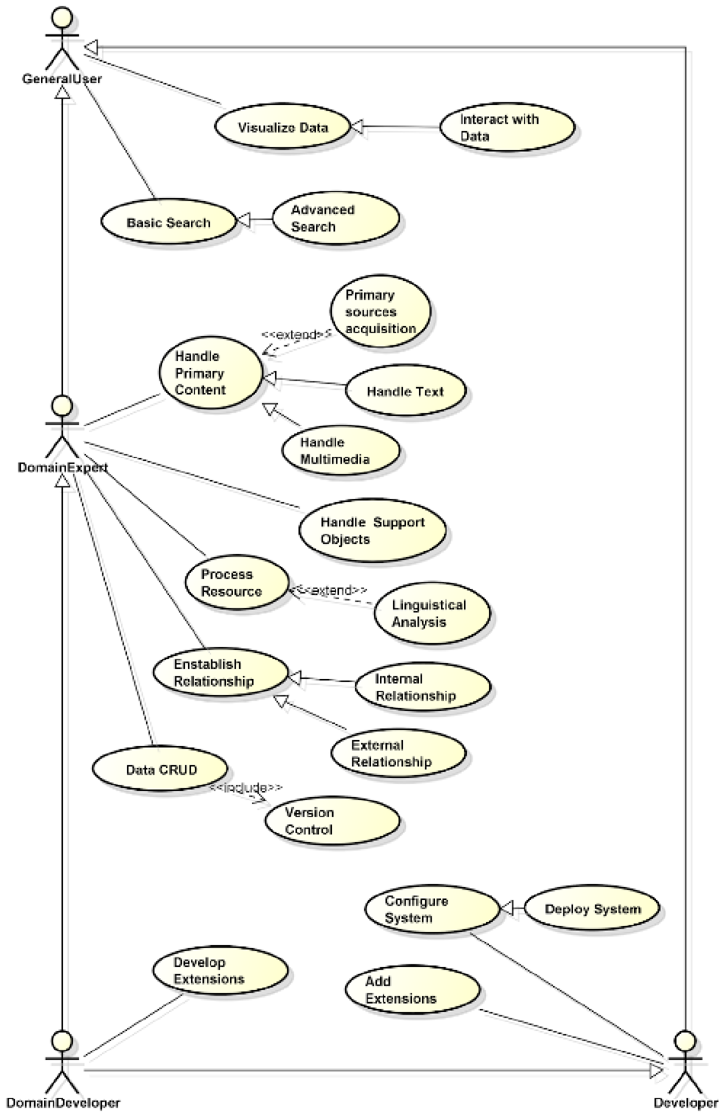


Figure 3.6: TSLib core Use Cases

Consequently, fundamental services emerge from the requirement gathering phase. They can be summarized as follows:

- Providing a textual scholarship workflow which starts with raw textual material. This can be an scan of a book or a picture of a manuscript. The workflow involves a collection of services available by means of Application Programming Interfaces (APIs) (local services by importing components or remote ones by calling Web services) [193]. The functionalities range from optical character recognition to text encodings, from named entity recognition to linguistic annotation;
- Providing tools for documenting (a) the textual choices adopted by the scholars, (b) the textual alternatives such as variant readings and collection of authoritative conjectures (c) the motivation for the text choice (d) the list of references on which the scholars work is based (e) the bibliography of editions and secondary literature (f) the commentaries of the works under investigation;
- Providing a publication system that supports the reading of the text in a multi-version manner [87, 6, 44];
- Providing facilities for scholars contribution. On the one hand, digital scholars can extend or reuse the software components, on the other hand thay can add or review textual data [175];
- Providing graphical interface for (1) web based applications, (2) mobile devices, and (3) desktop environment.

3.3 System architecture

In the digital era, the transmission and scholarly study of cultural heritage resources require technologies able to respond to literary and philological issues. Research activities where these kinds of systems need to be developed can be grouped into three blocks, which directly derive from the analysis phase (Fig. 3.7):

> **Acquisition and digitization of resources.** This module provides functionalities as Optical Character Recognition (OCR) of printed editions and

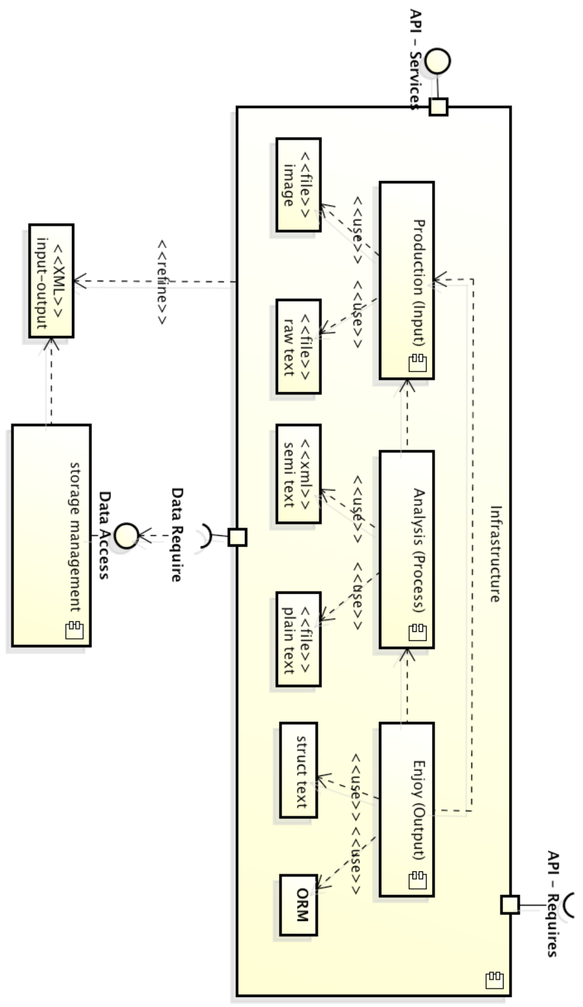


Figure 3.7: Textual Scholarship environment within a component schema

tools for document transformation from semi-structured resources to structured ones. This at least must take into account the systematic import of manuscript images and the XML-TEI encoding of non-standard rather than proprietary formats (e.g. RTF, DOC, etc) [194].

Content analysis and processing. This module is in charge of textual processing and document analysis, as well as Term indexing, information retrieval, information extraction and knowledge representation using ontologies. This corresponds to the multilayered indexing of the texts (linguistic, lexical, semantic and philological) combined with data processing and with the study of intertextuality; this in order to link different editions and entities to internal and external datasets.

Editing. This component deals with the design of Graphical User Interface (GUI). It handles resources in a distributed and collaborative environment (mostly Web-based). This allows scholars to collaborate in editing new editions or in proofreading automatic textual analysis. This means that the environment allows scholars to explore all the indexed texts and their different levels of analysis. Furthermore, it should provide access to a centralized repository containing: i) the translated editions and critical studies, ii) all the entities of interest belonging to different semantic classes cited inside the texts (people, places, events, etc.), and iii) comparative multilingual dictionaries. This module should also consider versioning issues.

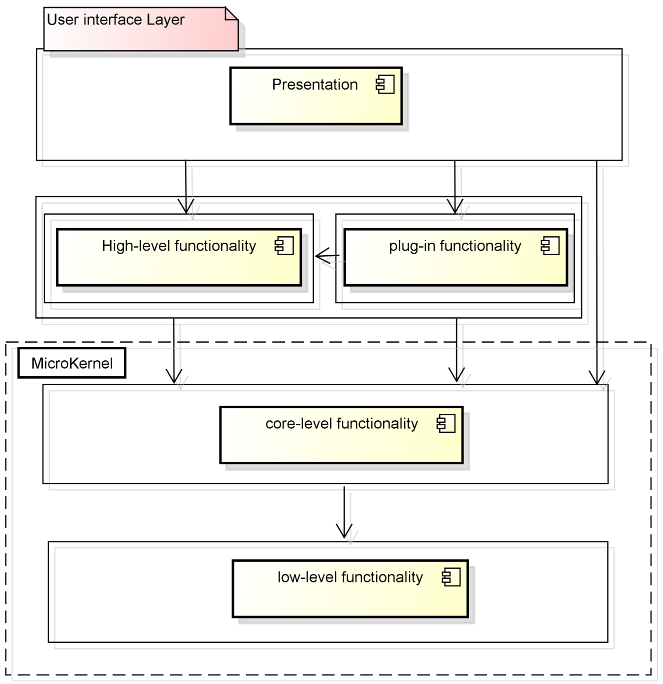


Figure 3.8: Layers view of the TSLib

In other words, the architectural design of the library for textual scholarship is designed to be modular and extensible and it is organized in different components upon different architectural levels (Fig. 3.8).

UML diagrams describe the component model and architecture. Each component emphasizes abstract interfaces which are both provided and required. In such a way, the provided interface defines the services that a component of the TSLib exports to its clients. Actually, interfaces may be seen as the entry points for controlling the component. Conversely, the required interfaces are functionalities that components expect to find from the external context in order to complete their own implementation.

TSLib is a collection of a number of subparts (rather than a monolithic entity) specifically designed to be embedded in a number of different contexts. Fig. 3.9 is a graphical representation of the library core components.

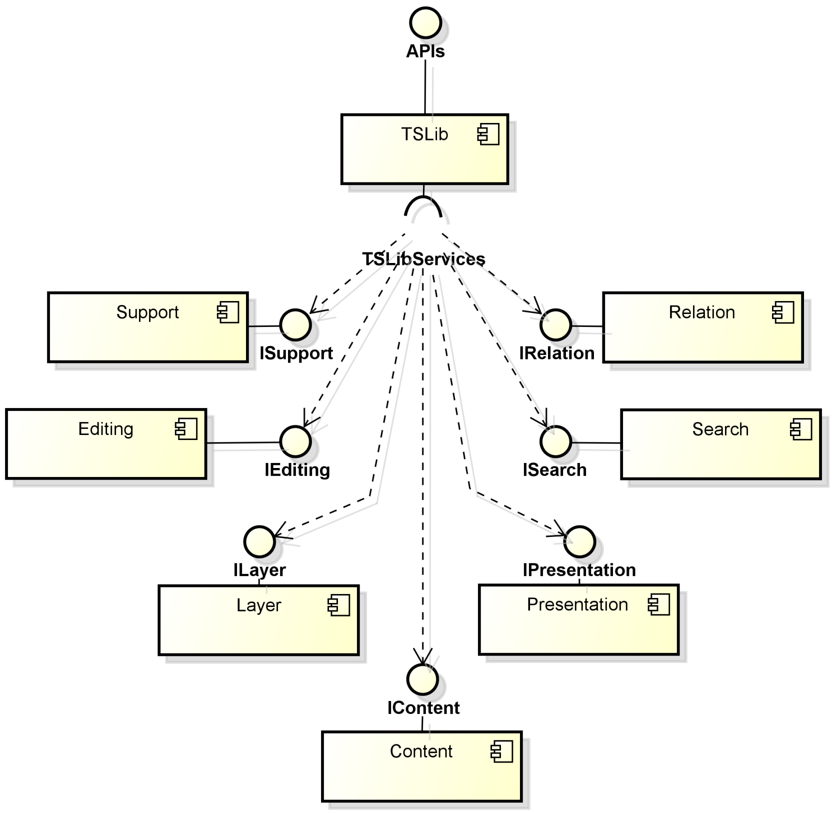


Figure 3.9: The core components of the Textual Scholarship Library

It shows the entry point and the subsystems composing the library. It represents a high-level view of the library and it is useful for understanding the general principles and aims the artifact deals with [195]. As the architectural image shows, the library focuses on provided and required interfaces. The components that provide interfaces act as servers, while the components that require interface act as clients. Thus, the TSLib interfaces define the library contracts and services. The required and provided library interfaces are public and they can be visualized as exported Application Programming Interfaces. These interfaces show the capabilities of the components and provide the methods needed to access the library by the client applications.

The core of the ongoing library is structured into several components which describe the top-level structure and the general organization of the library:

- Textual content management;
- Support/Facsimile management;
- Editing management;
- Layers management (analysis);
- Relations management (linked data);
- Indexing and search management;
- Presentation management (GUI).

The Textual content component handles the digital representation of the primary sources in an object-oriented framework. Section 3.4 widely illustrates the conceptual design of the textual entities, which efficiently meet the scholars needs. The model focuses on efficient access to different textual-phenomena at different levels granularities (e.g. page, line, word). The inherent complexity of the text structure is hidden from the conceptual data model and it can be implemented by modules that realize and extend the abstract classes.

The Support/Facsimile component deals with information related to the physical device [196]. These components manage the multidimensional models (e.g. 3D models, or a set of images of manuscript scans) of any relevant information related to a specific communication process of written text. It gives a complementary view of the digital representation of the textual document, by focusing on writing and context information that is not covered by computational linguistic processes. This is necessary for scientific interpretation and for understanding any text [197].

The Editing component manages the creation, the reading, the updating and the deletion of the data handled by the library. Moreover, the component has to preserve the integrity of the interconnected information, tracking multiple versions of the document, etc. Several types of textual objects are affected by editing, such as texts with variant readings, automated analyses, manual annotations and data entries for scholarly comments (annotation and linguistic enrichment are discussed in section 4.4).

Modules related to document and text analyses fall under the *Layer* component. These modules involve, for example, algorithms for lemmatization, pos-tagging, metrical analysis, named entities recognition, etc. These kinds of textual processes are pluggable extensions and can be customized both by service providers and clients by means of a specialized library module (see section 3.5.3). In general, the layer component concerns resources for their processing and annotation. The nature of these annotations leads to call the component "layer component". This means that documents convey data as different informative levels starting from the text itself (level zero) [198]. Other layers, built upon the text, can be phonetic, morphological, syntactical, etc. It is worth noting that the textual process has to be independent from the language as much as possible. This can be obtained by using *factories* and *delegation* patterns as illustrated in section 3.5.2.

The Relation component has been developed in order to handle the overall relationships among the entities. The relationships involve the digital entities through an identification schema (e.g. Resource Description Framework (RDF) within the LOD paradigm, see chapter 2 section 2.6.4 and 2.4.2). The linking is done at different levels of granularity and among different types of objects by means of a suitable hierarchical data model (see section 3.4). For instance, textual entities can be linked to other textual entities and a character can be linked to the related image box in its multimedia model.

The Search component creates and manages the data structures necessary to efficiently access stored resources (a practical case study is illustrated in Fig. 4.2). Search components, devoted to the information retrieval, combine the data indexed in the persistence unit and exploit a large number of query techniques for accessing databases (xquery, sql, sparql, etc.).

The Presentation component takes into account the data structures that represent content combined with multiple levels of analysis. Graphical interfaces, necessary for the interaction between the user and the system, must satisfy the scholars specific needs (user experience). In such a way, scholars avoid frustrations in using Graphical Interfaces designed for other domains.

The above introduced core components can also also adopted by international study groups for the development of an advanced textual environment (illustrated in Fig. 3.10), which aims at implementing a virtual framework for digital scholarly editing and for educational purposes [199].

The component interaction is guaranteed via ad-hoc mechanisms provided by specific manager modules [200, 201]. These mechanisms follow the service-provider framework [202] and allow multiple service providers to implement the same service. This method makes the implementation available to clients, while hiding the information related to the implementation itself. This gives great flexibility in choosing the actual module of the selected service. In such a way, the library can return object instances without making their class public. Hiding implementation classes leads to a very compact API and to interface-based frameworks where interfaces provide ADT thanks to static factory methods (see paragraph 3.5.2).

3.4 Designing the data model

The complexity, characterizing the textual scholarship domain, becomes more challenging due to the insufficient formal representation of sources from a philological perspective [50, 7, 11, 203]. As evidence, at present several researches are studying cultural data models (see chapter 2), but almost none of these embeds the vision that texts are transmitted by multiple supports, with multiple variant readings, and with multiple interpretations [204]. The object model here discussed shows hypothetical Abstract Data Types for the domain under investigation. Abstract Data Type (ADT) have been conceived as easy to reuse and as extensible for future improvements. Indeed, a data model that represents the conceptual object-oriented design of the artifact is an essential mean for the development of a suitable application.

The model relies on several object-oriented features such as encapsulation and composition. Moreover, the design tries to offer systematic solutions to recurring scholar activities. This configuration should serve as a guideline for the implementation an elegant and stable library for textual scholarship in an object-oriented

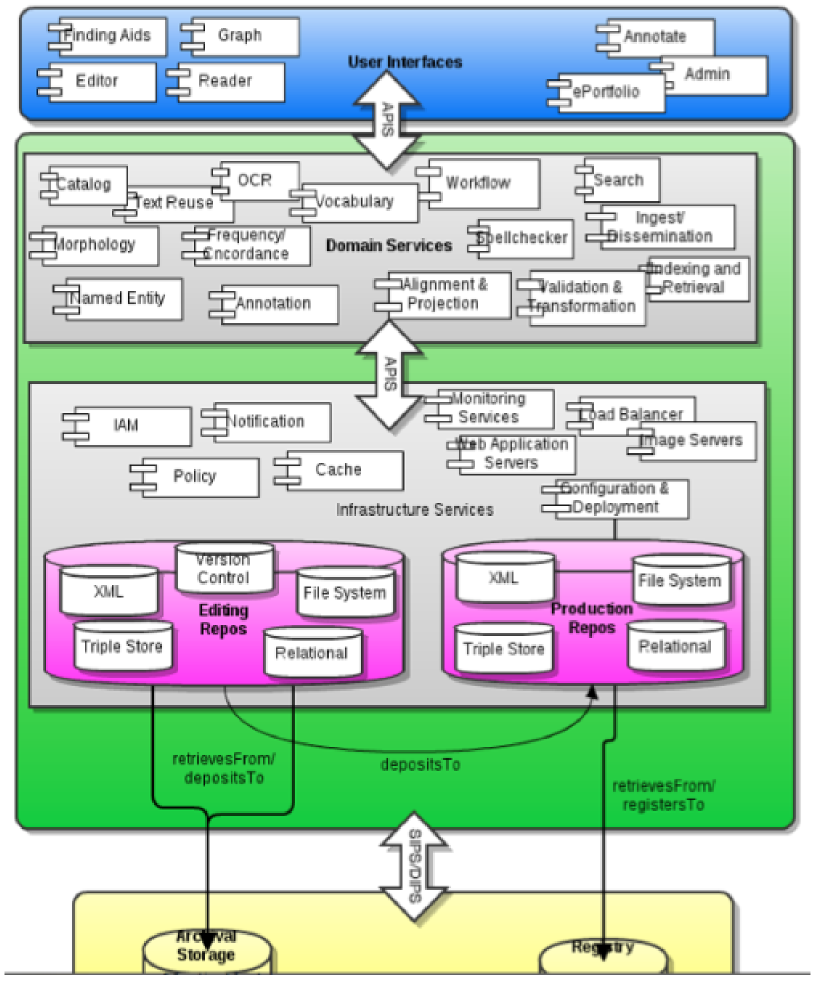


Figure 3.10: Open Philology architecture - Courtesy of Bridget Almas

programming style. One of the benefits of this approach is the ability to model the domain in terms of interconnected Abstract Data Types. In fact ADTs are domain entities representing the knowledge of the problem and they convey values and behavior that scholarly experts can understand and manipulate. This observation provides both conceptual and technical advantages [205].

Concerning conceptual advantages, documents and textual entities can be designed in terms of abstract objects through a platform-independent model. For example, a linguistic annotation or a comment are potential objects in the problem space of the library. As pointed out by [186], identifying core objects as data types in a given problem space and determining the object responsibilities and how they relate to each other is a preferable paradigm as compared to designing in terms of actions [206].

Concerning technical advantages, object-oriented approaches provide solutions as discussed in section 3.5.1. However, it is worth remembering that object modeling can lead up to deep inheritance hierarchies which can break the *Liskov's substitution principle* [207, 208]. Consequently, the design of the TSLib data model attempts to use inheritance only when appropriate, giving prominence to composition, which is often a more suitable solution [202, 209].

In the light of this, the key concept in modeling the literary domain involves how to design the information conveyed by each textual phenomena in a flexible way. As mentioned above, such a textual data encompasses (a) a multiplicity of variant readings, (b) different layers of analysis, (c) different interpretations, and (d) different levels of granularity. The model needs to reflect the conceptual requirement of the domain, rather than the actual software details. Following the emergence of new requirements, of functionalities and feedbacks, the model can change over time. For this reason, the software development process has adopted the Iconix-Agile process [185]. As discussed throughout the chapter 2, cultural heritage documents have complex content with multiple views and overlapping structures. This puts the accent on the need to further explore methods for their digital representation. Moreover, scholars need to identify the variants and manipulations to which manuscripts have been subjected. This allows them to discern trustworthy readings and accept or reject the hypotheses of previous editors. In addition, scholars need to examine the commentaries, articles and monographs concerning specific parts of the studied text. Therefore, the extension in *breadth* of the digital collections needs to be integrated by the extension in *depth*, according to the paradigms of the new generation of digital libraries [24, 210].

For these reasons, the design of the model should be based on four distinct concepts: 1) the textual structure; 2) the semantics of the structure; c) the style of the document; and d) the behavior of the entities. The points listed above ensure modularity, scalability and flexibility. The conceptual model of the library is shown through two class diagrams in Fig. 3.11 and Fig. 3.12.

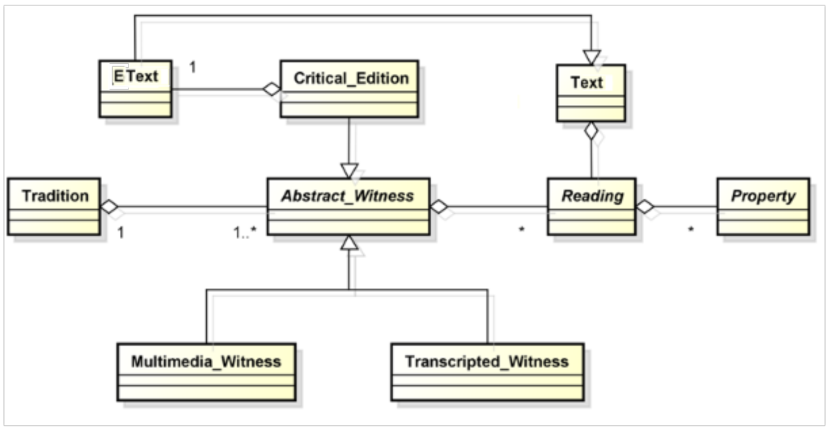


Figure 3.11: Class model defining the tradition of the textual documents

The entities define values and operations with respect to the requirements provided by the scholars. This means that the ongoing formalization of the domain tracks the model constraints and guarantees the consistency of the data properties. By doing so, the specification also determines the pre- and post-conditions of component behavior and their inter-module interactions.

The cornerstone of the model is the primary source representation. Documents which represent related sources have been managed in parallel through an abstract generalization (Fig. 3.13).

Annotating, commenting, and analyzing sources are some of the main activities dealt with by scholars. The two diagrams in Fig. 3.11 and Fig. 3.12 represent the central entities that the library handles in order to aid textual scholars in the comprehension of ancient, modern or contemporary documents.

The diagram in Fig. 3.11 describes the part of the data model which takes into account the **tradition** of the text, which is the *collection of witnesses* of the work under the study. The diagram also considers the reconstruction of the text

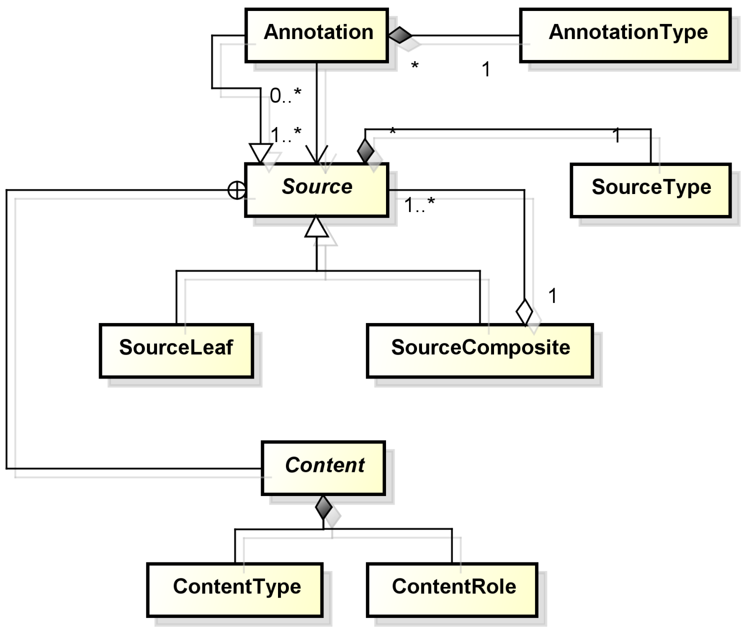
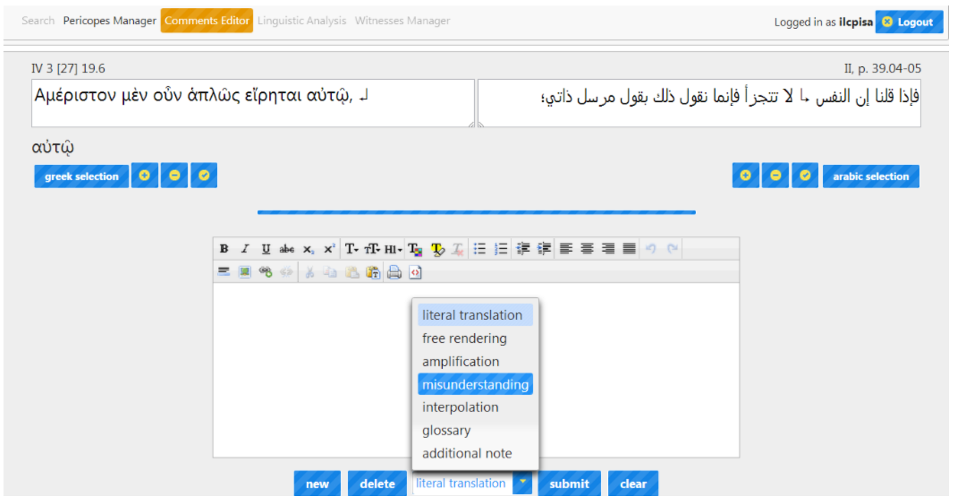


Figure 3.12: Conceptual Class Model representing textual source materials



IV 3 [27] 19.6

II, p. 39.04-05

Αμέριστον μὲν οὖν ἁπλῶς εἴρηται αὐτῷ, ↲

فإذا قلنا إن النفس ↵ لا تتجزأ فإنما نقول ذلك بقول مرسل ذاتي؛

αὐτῷ

Figure 3.13: Parallel bilingual texts with comments and selection

according to the editor, such as the **established text** in the **critical edition**.

The UML diagram formalizes the textual tradition, taking into account potential witnesses (text transcription and/or facsimile images) and **variant readings**. Variant readings ("Reading" in the diagram) have specific properties ("Property") which describe the nature of the variants themselves. Due to the multi-faceted nature of texts in the philological domain, the action of choosing a word acquires particular relevance. In fact, even in the case of documents (witnesses) showing the same text that they transmit, the possibility that errors have occurred throughout the history of the text is not excluded.

The diagram in Fig. 3.12 describes how the model represents the source documents. **Sources** are designed as a complex and *hierarchical objects* ("SourceComponent", "SourceComposite", "SourceLeaf") that can evolve in space and time ("SourceType"), creating variant readings ("Content") on top of which various kinds of **annotations** are set ("Annotation" and "AnnotationType"). Documents and textual entities are interconnected and structured objects, either of the same or of a different nature. These data structures and their interconnection might be expressed by linked data and made available for further analysis (see section 2.4 and section 4.4.2).

Fig. 3.12 shows the source class diagram. It describes the abstract entities that embody the multiple views and the hierarchy characterizing the structure of a resource from a philological perspective. Therefore, the abstract model has to consider the modalities and multiplicity of the resources. These can be realized through the introduction of a four-dimensional vector of properties <**v, g, p, l**>:

- the component "v" represents the version. It yields the information accessed by a version of the resource;
- the component "g" represents the granularity of the object. It conveys the hierarchical, structural representation of the resource (see Ordered Hierarchy of Content Objects (OHCO), Functional Requirements for Bibliographic Records (FRBR), Canonical Text Services (CTS));
- the component "p" describes the position of the textual object. This means for example whether it is a first edition, or the second page, or the third paragraph, or the fourth sentence, or the fifth word, or the sixth character;
- the component "l" introduces the layer of analysis. It is responsible for representing different levels of interpretations, such as morphological tagging, syntactic parsing, semantic annotations, metrical analysis, etc.

The above diagrams shape the principal concepts that the library has to manage. These are: a) *the hierarchical structure* of the sources, expressed in the diagram by the composite pattern (see section 3.5.2); b) the *multiplicity views* of the content, expressed by different roles and different typologies (relationship typed pattern); c) the *external annotations* upon the sources.

The aforementioned framework models the basic relationships among the objects by following the nature of the target domain. This is outlined as:

The source data type is specified by two sub-type entities. The composite relationship allows to design a hierarchy between the various levels of the document and the textual elements. This solution is suitable to express levels of granularity and non-linear structures. For example, the textual content of a document can be organized in pages, lines, words, characters, etc. This representation is hierarchical and each element can be seen either as a new composite element or as a leaf element.

The source entity comprises various nested data types generalized by a common abstract class. This relationship shapes the multiplicity of views of the textual content. For example, a textual block can have a correspondent digital representation (usually an image). The type of data content defines the aspects in a document for what relates to *metatext*, *paratext*, and *extratext*, by means of a typed relationship pattern (**ContentType** and **ContentRole**)[4].

The annotation concept draws up a stand-off mechanism. It combines multiple levels of analyses and comments in order to enrich the information corresponding to the target source. Generally, annotations can be seen as metadata, that is to say, information that provides contextual data about an object in the collection (see chapter 2). Metadata can also be viewed as primary data sources that are related to or derived from other primary data. Each annotation is endowed with a specific type. This technique provides extension purposes and flexibility. For example, automatic linguistic analysis could add *lemmata* to each word, but scholars could add exegetic comments to some textual paragraphs. References to annotated entities (as CTS Uniform Resource Names (URNs) do) allow the handling of related sources for human and for software agents. Consequently, this facilitates the

[4]Further information on text, paratext, extratext and intertext can be found in literature, for instance [211, 25, 3]

alignment of different texts versions with the relative annotations. As discussed in section 3.5.2, the *observer pattern* implements change-listener mechanisms that track the various updates brought to the content. These mechanisms manage the notification of annotations related to source objects even when a version is removed or modified at any level of granularity. Finally, the Open Annotation Data Model (OA) and CTS URNs help managing and encoding annotations that point to the source textual content. The digital representation of textual content with the addition of annotations fosters a hierarchical graph representation [87], instead of a tree representation.

In summary, the data model described above provides the core entities regarding the digital representation of textual documents within a scholar perspective. It allows to manage hierarchical views of the content as a flexible relationship among texts, images and other textual types such as "paratext" and "extratext". In addition, it also automatically handles all annotated contexts.

As discussed in section 3.5, design patterns [212, 213] are the means to organize modeling efforts into classes, methods, and objects. These objects store data and provide methods for accessing and modifying their own internal state.

3.5 API design and Design Patterns

The TSLib sets its design both on API techniques and Patterns. As already pointed out, the novelty of this research lies on the application of these techniques to the domain of literary computing.

Hereafter, examples regarding the design principles and techniques that cover diverse computational themes are introduced. The diagrams illustrate design choices for performing textual services. These latter can include the identification of a word as a noun, verb, as well as metrical analysis services, identifying rhythmic structures in a text. They also can include lemmatization capability, which identifies the basic word-form in the text representing the entire word paradigm.

This choice is justified by the fact that the library combines experience, methods and tools developed by the Institute for Computational Linguistics (ILC) which invests in computational solutions both for the fields of linguistics and philology [214].

3.5.1 API design

This research has investigated well-known and fundamental strategies and principles to achieve abstract designs and effective API development. As already stated, the relevance of this topic stands in its potential to design and release a reusable library. As it is commonly acknowledged, the purpose of an API is to separate client applications from the providers that implement the services. It derives that API users do not know the details of the artifact, as they only have access to public information. Furthermore, as providers work does not take clients into account, the former can change implementations without breaking the interfaces. Thereby, APIs enable TSLib to define and develop components separately for the various features that make up the library. Based on this, it is possible to reuse and extend the artifacts to a range of applications and in different moments. In addition, this approach allows to solve problems in classes, rather than individual problems, keeping clients and providers separated. Indeed, if the client is only acquainted with a clear specification (contract) between the API and the implementation (e.g. the behavior), providers have to comply to the general contract, but they are also entitled to change the underlined algorithms [209]. Eventually, the objective is to allow the components to evolve without forcing clients to change their code.

Against this scenario, designing a modular library for literary computing implies, among other things, a deep understanding of general API features, such as abstraction, inversion of control, code reuse, open-close principle, and loosely coupled components. These factors contribute to the implementation an appropriate and efficient tool, tailored on computational scholars needs. It derives that application Application Programming Interfaces design can affect the behavior, the soundness, and difficulty of using software components [195]. As a matter of fact, when an API changes, all clients and implementations have to change. Therefore, the discussion about the Application Programming Interface cannot be limited to the naming of basic object classes, or public methods exported to the users (in other words, an API does not end at the *signature* of a class or method [202]). Such a design, instead, is much more challenging since APIs encompass more complex issues:

- specifying information that scholars need to know in order to use the library;
- fostering scholars to reuse modules having restricted knowledge of the API;
- avoid making all implementation information explicit to users, in order for

making; an intuitive exploration of the library (*Shallow Understanding*);

- assembling individual building blocks in order to generate the whole application;
- evolving compatibility for anyone writing applications based on the library;
- being as small as possible but not any smaller;
- helping providers to change the code and therefore implementing different; algorithms for more efficient and more accuracy problem solving;
- specifying the set of protocols, files, environmental variables and their formats, that each component of the library must read or write;
- ensuring that the behavior of a component remains unchanged.

At this stage, the computational scholars' attention turns towards the programming code that they need to write in order to accomplish their tasks. In fact, APIs are user interfaces for computational scholars just as GUIs are user interfaces for classic scholars. In such a way, APIs allow programmers to ignore what is the actual data representation and what are the details of the implementation algorithms. In this way, API establishes an abstraction over the functionalities of each component and hides the unnecessary internal complexity. According to this view, an Application Programming Interface is a contract [215] between the programmer and the implementation providers, as well as between the API designer and the developers of the specific functionalities. In accordance with this, the challenge resides in designing clear and unambiguous contracts and functionalities for the library. In order to clarify this concept, hereby an example of a Java code snippet is given. The client of the library instantiates the object, which represents a manuscript, and associates the facsimile images for each folio. Afterwards, he asks for the textual content of the document specifying the data format. In the end, the client writes the data into a stream either for storing the outcomes or for communication purposes through a socket. The client is unaware of the implementation mechanism and consequently, the procedure appears for him as natural and intuitive as possible. In fact, the objective of a well-defined API is to provide a logical interface to the exported functionality of a component while hiding any implementation details [216, 217]. In the program code lines below, error checking and exception handling have been omitted in order to make the example clearer.

```
Source codexM = Source.newInstance();
codexM.setMetaData(properties.get(‘‘nfolia’’));
codexM.setFacsimile(properties.get(‘‘inputSource’’));
String jsonText = codexM.getText(Granularity.FOLIO, Format.JSON);
write(jsonText);
```

In the light of what has just been argued, firstly, any tool for scholarly studies should provide an abstraction for the needs of historical and cultural documents. Secondly, it should specify the guidelines according to which scholars should interact with the components of the library. In turn, each component should provide solutions to the basic issues of the domain, such as annotations of textual phenomena, correlations between text and images, highlights of formulae and linguistic patterns in different languages, alignment of different versions of the same text, semantic searches based on shared ontologies, scholarly editing, textual criticism, and variant reading management. Essentially, APIs define reusable bricks that allow modular pieces of functionality to be imported into the end-user applications. For example, Fig. 3.14 describes a typical text analysis use-case where a client application handles lemmatization and concordances. The application depends directly on several modules by means of exported APIs.

As mentioned before, a primary goal of this research is to examine the fundamental characteristics that compose a sound tool in the field of literary computing. This implies that the library meets some design qualities [190, 195], which are often disregarded by the software within the Digital Humanities, but that are desirable, when implementing a software that is to be reused. The most outstanding qualities adopted for the TSLib can be listed as follows:

- Information Hiding [205, 218],
- Comprehensibility [191, 219],
- Consistency [202],
- Discoverability [219, 220],
- Difficulty to misuse [221, 222, 187],
- Performance [223, 224],

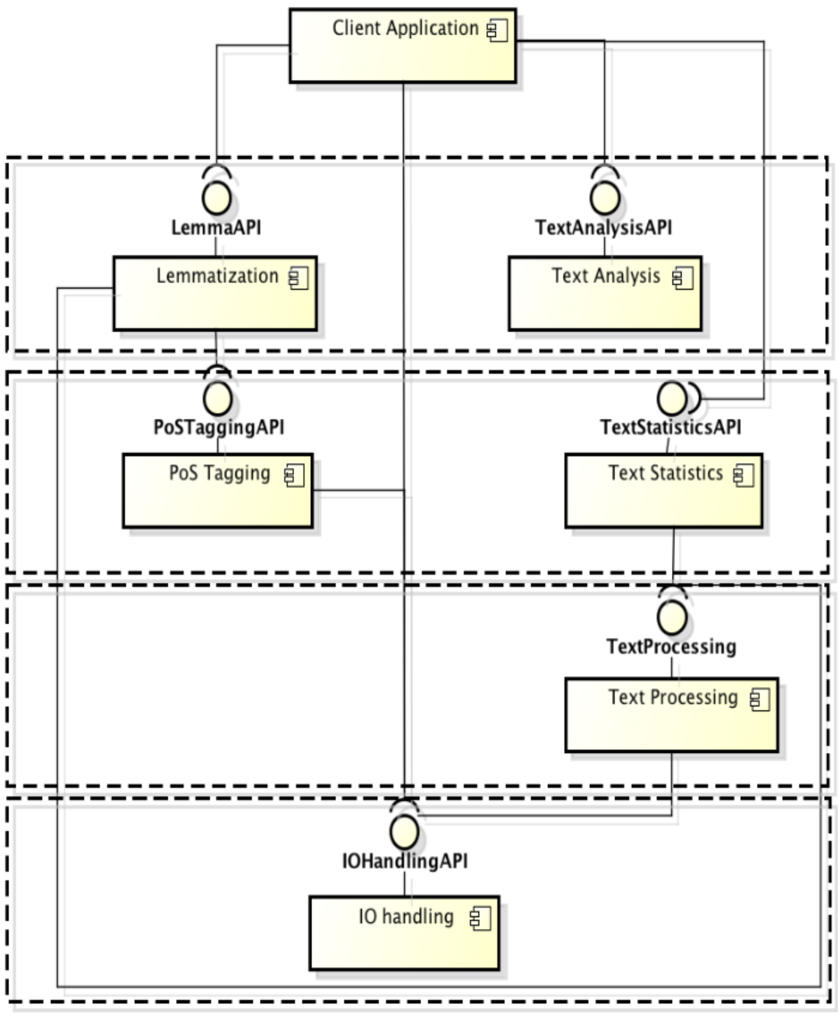


Figure 3.14: Lemmatization example for API Component design

- Stability (Backward Compatibility) [202, 225, 226].

Information Hiding produces reasonable advantages in terms of (a) performance (i.e. it enables caching techniques, lazy evaluation, etc.), (b) resilience (i.e. it forecasts validation, notification, synchronization mechanisms and allows the environment to maintain invariant relationships and verify pre- and post- conditions), (c) evolvability (i.e. improving implementations by adding new functionalities through plug-ins). Design patterns contextualized to the target domain (see section 3.5.2) guarantee a less-coupling modular system and a separation of concerns.

Comprehensibility means that computational scholars must understand the core model of the library, its key objects and functionalities. The principle to be followed is called *least astonishment* (aka least surprise). Hence, API is to be understood as a meta-language that involves three actors: (1) the designer of the library, (2) the providers which write implementations against the API, and finally (3) the clients that write custom applications. These actors communicate thanks to the availability of the API.

Consistency deals with mechanisms that handle similar features through similar ways. It is important that these kinds of mechanisms follow the same policies across the components. For instance, if the policy of passing arguments to methods were performed via special Data Transfer Objects (DTO) together with String-typed values instead of enum-typed values, these policies should be coherent throughout the entire procedure. The following is an example of code involving text analysis:

basic method:

```
ResourceAnalyzed analyze(SourceDTO source, String action)
```

alternative method:

```
ResourceAnalyzed analyze(SourceDTO source, AnalysisAction action)
```

calls:

```
SourceDTO source = FactoryDTO.instanceSourceDTO();
source.setContent(content);
source.setParameters(parameters);
```

basic call:

```
ResourceAnalyzed object = analyze(source, ‘‘morpho’’);
```

alternative call:

```
resourceAnalyzed object =
analyze(source, AnalysisAction.morphological);
```

Discoverability deals with organizing entry point classes as intuitively as possible. It is in charge of creating a single place that can serve as a starting point for discovering the component services. For instance, the well-known NetBeans integrated development environment (IDE) platform[5] organizes lookup mechanisms to manage services and discover registered functionalities [52, 227]. In this context it is important to provide examples of API employment, aimed at accomplishing specific tasks.

Difficulty to misuse involves some other library design features such as minimal complete, clear, simple, intuitive, easy to memorize, and other. The rule, which guides the design, is that the API ought to be for computational scholars like the GUI is for classical scholars. It follows that one of the most decisive points at issue is naming. Few valuable guidelines are: (1) avoiding abbreviations, (2) guiding user for methods parameters, (3) avoiding long list of parameters with the same type (4) adhering to naming conventions. The example below shows these rules applied for the design of the class that represents the morphological code in the library.

poor signature:

```
Public MorphoCode produceMorpho(
String pos, String per, String num, String tense, String mood,
String voice, String gen, String case, String deg);
```

difficult call:

```
MorphoCode mc =
MorphoCode.produceMorpho(‘‘a’’ ,‘‘-’’,‘‘s’’,‘‘-’’,‘‘-’’,
‘‘-’’,‘‘m’’,‘‘a’’,‘‘-’’);
```

better signature:

```
Public MorphologicalCode
newInstance(
PartOfSpeech pos, Person p, Number n, Tense t,
Mood m, Voice v, Gender g, Case c, Degree d);
```

easy call:

[5]ttp://www.platform.netbeans.org

```
MorphologicalCode mc =
MorphologicalCode.newInstance(
PartOfSpeech.ADJECTIVE,
Person.NAN,
Number.SINGULAR,
Tense.NAN,
Mood.NAN,
Voice.NAN,
Gender.MASCULINE,
Case.ACCUSATIVE,
degree.NAN);
```

Performance A well-designed library usually results in a good performance. Generally loosely coupled components enable effective performance tuning. Indeed, once a system is complete and profiling has tracked which modules suffers in performance, those part of the system, can be improved without affecting other modules.

Stability mainly concerns backward compatibility. The library has to achieve evolution and enhance features without breaking old applications. This means that the library has to improve without affecting client code stability. The APIs, as pointed out in [202], manifest multiple levels of backward compatibility: (a) Source Compatibility, (b) Binary Compatibility, and (c) Functional Compatibility:

- *Source Compatibility* deals with the ability to compile client applications against new library releases. The challenge involves adding both methods and classes.
- *Binary Compatibility* deals with the ability to link new releases of the library against client applications without recompiling them. The binary compatibility fosters the use of public methods instead of static or public fields, in order to manage compatibility by dynamic bindings.
- *Functional Compatibility* deals with component-based development. This means making the modules behave the right way across library releases. This kind of compatibility relates to specifications: the more complete and

comprehensible the specifications are, the more functional compatibility is expected. The risk is that users rely on an unexpected functionality.

It is evident that the aim of the library APIs is to hide any implementation detail and foster modular programming. By doing so, the evolution of the library does not affect existing application clients. This means that any internal detail and changing element must be kept hidden from the client of the API.

In addition, the library benefits from *generic feature capabilities* [228, 229]. In this regard, Fig. 3.15 shows the aligner component which puts into effect this aptitude.

Actually, generics allow clients to write custom applications in terms of generic types. In the above mentioned example, scholars specialize the generic definition of the aligner entity by instantiating it with specific types. This is instrumental for the aligner to use generics to handle sequences of objects of any type.

APIs, which follow the aforementioned principles, require clients of the library to write few lines of code to perform basic tasks. Meanwhile, the software package allows clients to control the processes they want to accomplish. High-level convenience components [205] provide a well-known solution to wrap core APIs and, consequently, give functionality on top of the basic one. This solution, at the same time, guarantees great flexibility of usage for complex tasks and the least possible efforts for simple activities. These convenience wrappers do not depend on internal methods or symbols of the library. In fact, they are fully isolated from the core API and rely only on the public interfaces of the core components.

The textual scholarship library should prevent its clients from breaking encapsulation. Generally, to face this behavior the exported objects are to be declared *final* and *immutable* [209, 202]. When appropriately designed, a core API yields applications less prone to errors and they are more secure. For example, the access to any mutable components has to be exclusive. At the same time, as claimed in [209], no objects which have fields referring to mutable data should provide references to the latter mutable fields, either in the initialization phase or when returning the objects.

Hence, it is important that internal classes, interfaces, and members are not part of the API. This practice is largely used in API design and it is known as functional approach because a service does not modify any operands but it returns data only by manipulating new structures. In this way, immutable objects can be shared freely. Furthermore, it is easier to maintain the invariants of a complex

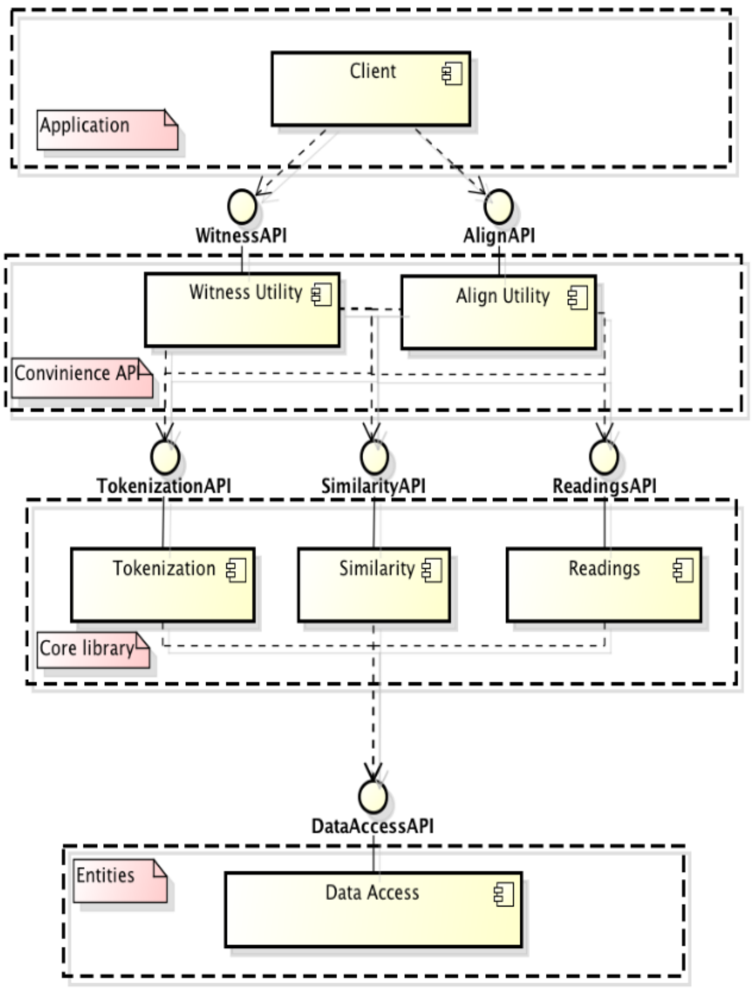


Figure 3.15: Convenience layered API for TSLib

component if the entities it handles do not change. However, in order to keep the quality of performance high, some immutable classes have one or more non final fields in which they can cache the results of expensive processes.

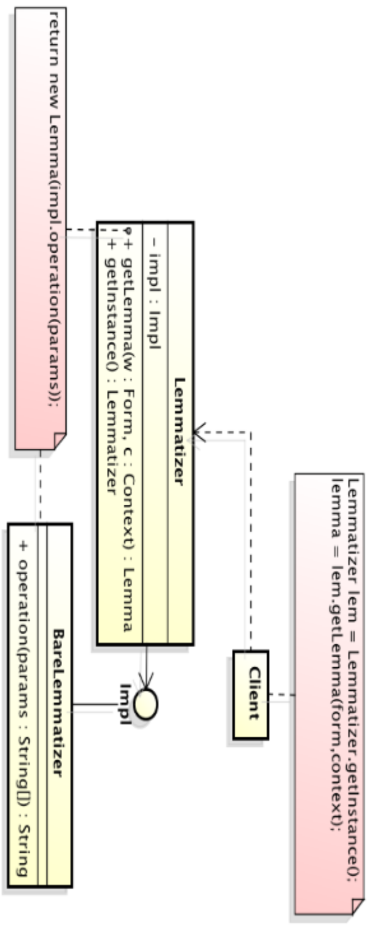
In conclusion, the API approach entails three main objectives. First, it enables the production of a clear and correct client code. Second, it fosters the development of prototypical implementation against the core functionality. Third, it promotes to implement end-user applications to validate the design decisions.

3.5.2 Design Patterns

The design of the TSLib makes use of common design patterns [212, 213, 206, 230]. Hence, the work has investigated software design and experimented various types of patterns, techniques and principles popular in software engineering. Indeed, one of the aims of this work is to apply design patterns that represent general reusable solutions to recurring problems within the domain of literary studies. The main patterns and techniques effectively adopted in the design of the library are explained below.

API/SPI delegation The technique that allows the library modules to clearly decouple client modules from provider modules is the API/Service Provider Interface (SPI) delegation technique. SPI stands for Service Programming Interface and should be completely separated from client calls [52]. Fig. 3.16 shows how this technique is able to hide internal details from public exported methods.

That is, moving data representation and current algorithms to a customizable and private class with implementation purposes. This solution allows to decouple APIs from their provider modules (SPI). This pattern has many useful advantages. The separation of public interfaces from implementation details also means that the client API can evolve, differently from internal interfaces which are only suitable for those who implement the services. As a matter of fact, users interact with objects that are stable and final (in the example, the **Lemmatizer** class). Anyway, the providers' perspective keeps flexibility for the implementation of the required interfaces (in the example, the **Impl interface**). In such a way, the library has a reliable mechanism for extending its functionality by improving the implementation and by adding new interfaces in further releases. When new features are added, the exported objects remain unchanged thanks to a particular delegation technique (i.e., via *factory methods*, see paragraph 3.5.2). *Delegation* is a powerful strategy that allows service modules to cooperate with the clients. The API/SPI pattern fosters binary compatibility: the reference to the implementation interface does not change in the subsequent versions of the module. Furthermore, the introduced mechanism guarantees performance tuning about system properties and resources availability, as it is possible to *lazily allocate* the internal class or to decide which type of object to allocate.


Lemmatizer
– impl : Impl
+ getLemma(w : Form, c : Context) : Lemma
+ getInstance() : Lemmatizer
BareLemmatizer
+ operation(params : String[]) : String
Impl
Client

```
return new Lemma(impl.operation(params));
```

```
Lemmatizer lem = Lemmatizer.getInstance();
lemma = lem.getLemma(form,context);
```

Figure 3.16: API/SPI Delegation pattern

Factory methods Fig. 3.16 also shows the *factory method* pattern. This is a wide used and a well-known creational pattern which provides a flexible way to instantiate an object. Factory methods are frequent in basic abstract data types across the design of the TSLib.

The main advantage of factories is to allow clients of the library to create objects without having to specify the actual type. For example, in Fig. 3.17 which shows the class diagram describing the sequence alignment module of the library, factory objects (**AlignerFactory** and **SimEvaluatorFactory**) create the actual entities to perform the aligned task.

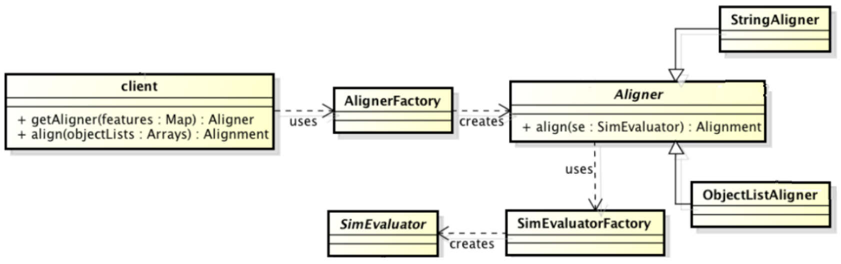
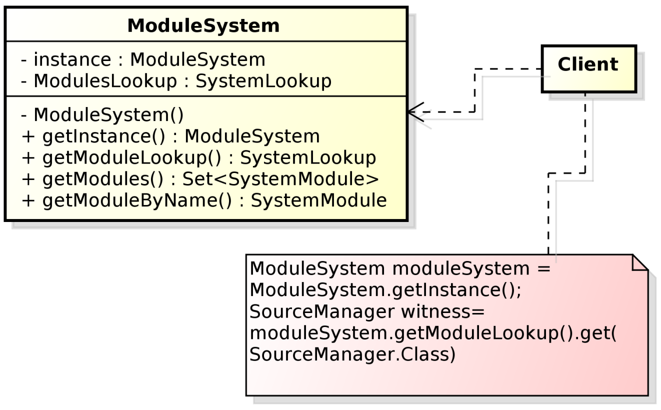

client
+ getAligner(features : Map) : Aligner
+ align(objectLists : Arrays) : Alignment
uses
AlignerFactory
creates
Aligner
+ align(se : SimEvaluator) : Alignment
StringAligner
ObjectListAligner
uses
SimEvaluatorFactory
creates
SimEvaluator

Figure 3.17: Factory Pattern

The client of the module uses the factories having no awareness of what the actual instance objects are. The client, in fact, just invokes interface operations implemented in all sub-typed classes. As the design of the aligner module demonstrates, it is possible to achieve more flexibility during construction procedures by adopting factory strategies. This moves object binding at run time, rather than at compile time, as required by conventional constructors. This behavior is largely adopted in designing the library since public classes can create different objects based on user input (the *getAligner* method in the example supplies a map of features). Furthermore, actual objects can be created based on configuration files or context properties available at run time. In this way, scholars can ignore the internals and the specific types of the different aligners.

Data driven Fig. 3.14 shows another technique known as the Data-driven approach. Throughout this solution the **Lemmatizer** class can perform different operations by supplying the delegated object with different parameter data. By doing so, the public methods guide the external client with methods which should be as clear as possible. instead, internal mechanisms can provide more generic routines that accept method parameters with a named-command idiom. It derives that, the implementing modules can perform different operations without requiring the software to be recompiled. Moreover, the modules benefit from an improved backward compatibility because the operations, the commands and the arguments are encapsulated in the data-driven model.

Singleton Fig. 3.18 illustrates the *Singleton pattern.* The component that manages the TSLib modules is designed to manage the load/unload functionality for the set of active components of the library. In particular, the component exploits the *Singleton* pattern as well as the *Lookup* techniques following the principles expressed in [205, 202]. This pattern supports the creation of a singular and global instance of the Manager object. Consequently, concerning the execution context, the library counts on one module which is in charge of managing the remaining modules.


ModuleSystem
- instance : ModuleSystem
- ModulesLookup : SystemLookup
- ModuleSystem()
+ getInstance() : ModuleSystem
+ getModuleLookup() : SystemLookup
+ getModules() : Set<SystemModule>
+ getModuleByName() : SystemModule
Client

```
ModuleSystem moduleSystem =
ModuleSystem.getInstance();
SourceManager witness=
moduleSystem.getModuleLookup().get(
SourceManager.Class)
```

Figure 3.18: Singleton Pattern

Proxy, Adapter and Facade The TSLib takes advantage of existing pieces of software following the principle of code reuse. The structural patterns like *Proxy*, *Adapter*, and *Facade* allow to design mechanisms for wrapping components on top of other APIs.

Fig. 3.19 shows the **TSLibPosTagger** on top of the *OpenNLP PoSTagger*.

Linguistic analysis of the library benefits from robust and efficient third party libraries. However, this pattern keeps the consistency of the textual scholarship library allowing its providers to export their own method signatures. Fig. 3.20 shows the Adapter pattern.

Similar to the proxy pattern, it provides a one-to-one mapping of new functions to pre-existing operations, but the interface and the APIs are different. This pattern is useful for exposing suitable APIs from existing legacy modules. The figure illustrates how the TSLib API is able to maintain its consistency from the client perspective, while internal procedures reuse pre-existent software modules.

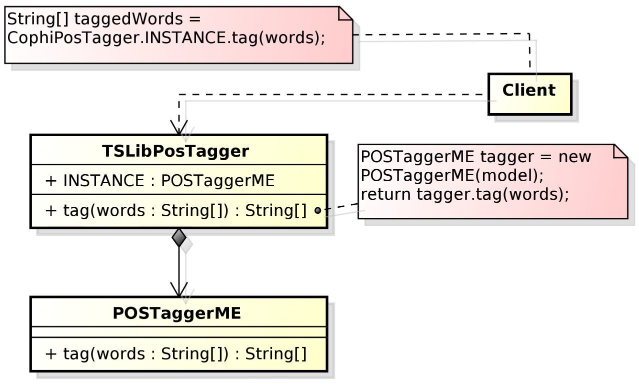

String[] taggedWords =
CophiPosTagger.INSTANCE.tag(words);
Client
TSLibPosTagger
+ INSTANCE : POSTaggerME
+ tag(words : String[]) : String[]
POSTaggerME tagger = new
POSTaggerME(model);
return tagger.tag(words);
POSTaggerME
+ tag(words : String[]) : String[]

Figure 3.19: Proxy Pattern

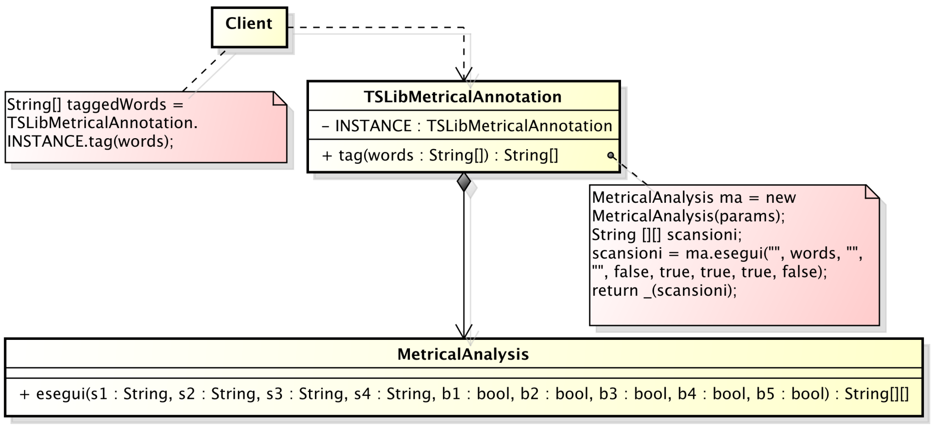

Client
String[] taggedWords =
TSLibMetricalAnnotation.
INSTANCE.tag(words);
TSLibMetricalAnnotation
– INSTANCE : TSLibMetricalAnnotation
+ tag(words : String[]) : String[]
MetricalAnalysis ma = new
MetricalAnalysis(params);
String [][] scansioni;
scansioni = ma.esegui("", words, "",
"", false, true, true, true, false);
return _(scansioni);
MetricalAnalysis
+ esegui(s1 : String, s2 : String, s3 : String, s4 : String, b1 : bool, b2 : bool, b3 : bool, b4 : bool, b5 : bool) : String[][]

Figure 3.20: Adapter Pattern

As argued in the Introduction, this kind of pattern is especially useful when the design adopts a trade-off between a top-down and a bottom-up approach. Fig. 3.20 shows the metrical analysis Adapter[6]. Nevertheless, the textual library takes advantage of the metrical module as one of the layers of textual analysis without exposing the original interface and the data types.

[6]The original software has been developed within the Musisque Deoque and the Memorata Poetis projects [231], but APIs and Abstract Data Types are not compatible with the design of the library

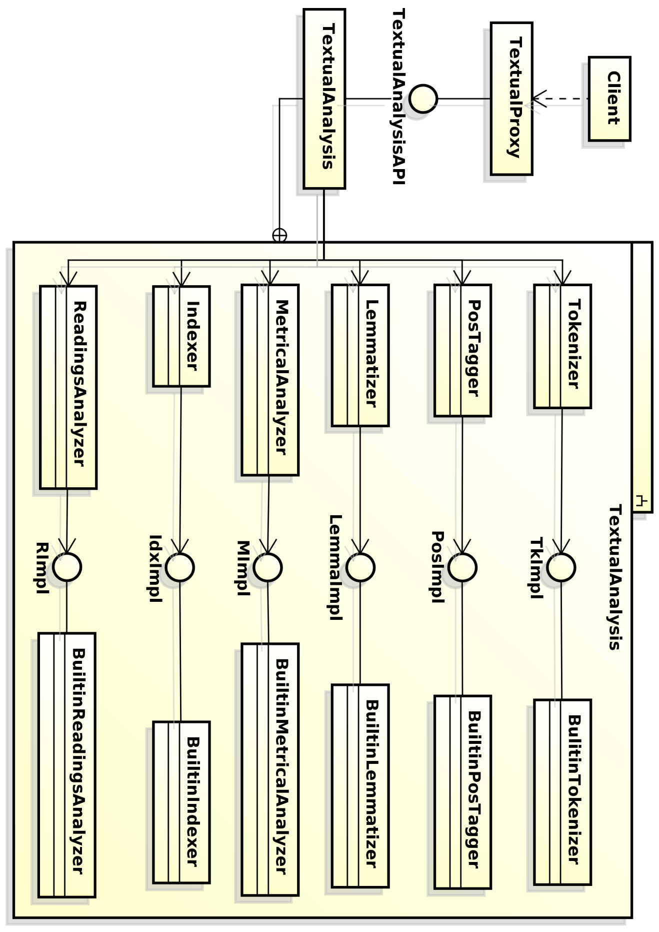


Figure 3.21: Facade Pattern

High-level components and library functionality exploit abstract access points for managing large collection of sub-modules. In order to achieve this ability, the *Facade pattern* has been adopted within the core component design. This pattern allows to split the artifacts into easier parts (top-down approach) and consequently it fosters stable APIs on top of a variety of entities. Fig. 3.21 illustrates these concepts with respect to tokenization, lemmatization, metrical analysis and indexing.

The facade pattern is often adopted in synergy with other patterns: the figure shows the Factory, the API/SPI, the Proxy and the Adapter working together.

To sum up, the TSLib makes full use of structural patterns as they present many benefits, ranging from performance to extension capability.

Composite pattern As highlighted from OHCO model (see chapter 2), textual documents have hierarchical structures. In the context of the object oriented paradigm, the Composite pattern shapes part-whole hierarchies in a tree manner. Fig. 3.22 shows the solution adopted for representing the structure of the document. Any primary resource in the TSLib model is a **Source Component** object which can be a final node (**SourceLeaf** entity) or an aggregation node (**Source Composite** entity). The figure also shows the typed relationship pattern, which provides flexibility to the **SourceComponent**: The **SourceType** indicates the nature of the node. This pattern allows to manage any kind of sources.

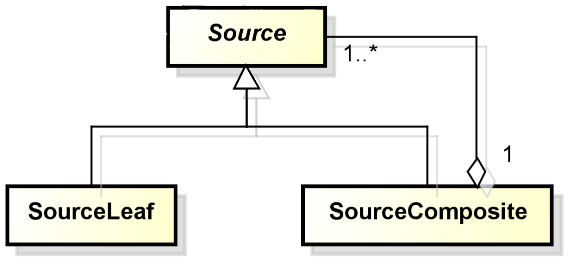


Figure 3.22: Composite Pattern

Observer, Strategy, and Visitor Classical design patterns include behavioral techniques such as *Observer*, *Strategy*, and *Visitor*: the TSLib has adopted these patterns to reduce strong dependencies among the modules of the library. For instance, the Observer pattern allows entities to communicate by means of a single point of anchoring, which provides the exchange and the delivery notifications.

In particular, as illustrated in section 3.1 and 3.4, the digital representation of cultural heritage documents needs synchronization and updated mechanisms [232]. Indeed, document components encompass an interdependent collection of formal annotations and interpretations that have to be kept updated and synchronized [233, 234].

For example, Fig. 3.23 shows how annotations concerning textual data can be notified and updated if the underlying text changes. In this case *the subject* of the pattern is the text of the document and the *observers* of the pattern are the annotations. The entities representing the text provide subscription and notification mechanisms. Furthermore, the observers, namely the annotations, comments, and external data, implement an update method in order to re-synchronize their state. In this way, whenever the representation of the document changes or any internal data of the document is modified, a notification of the change is sent to the affected entities.

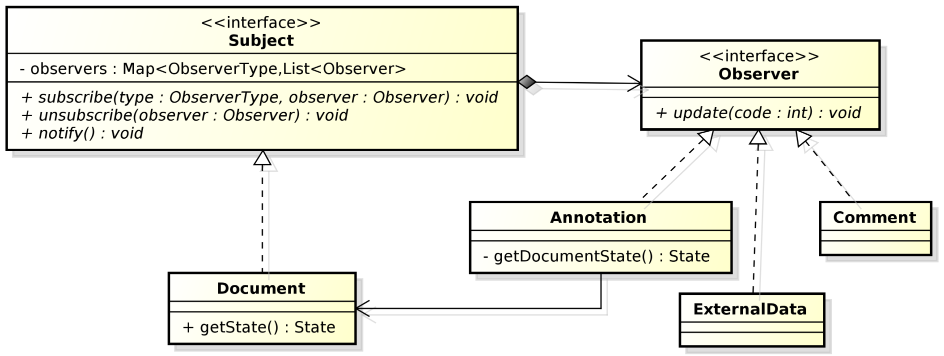
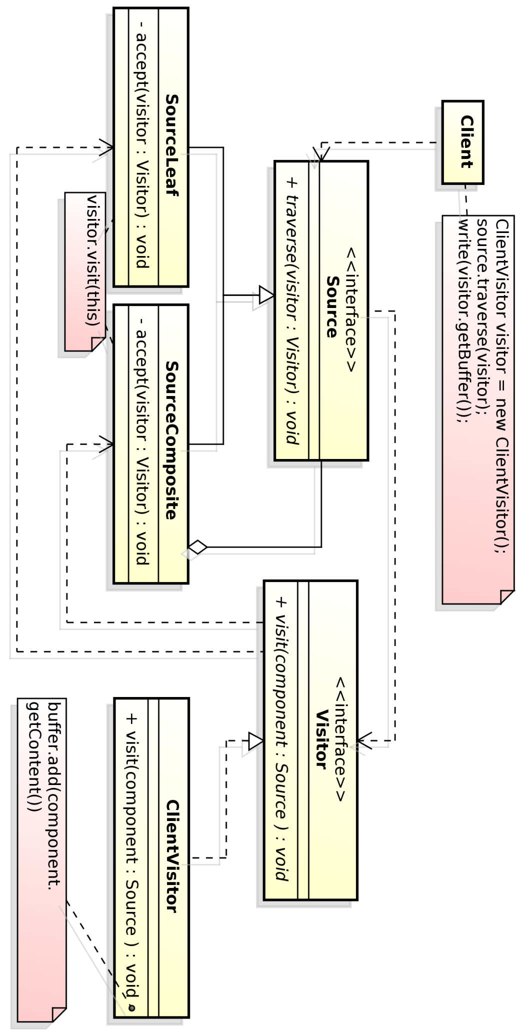

<<interface>>
Subject
- observers : Map<ObserverType,List<Observer>
+ subscribe(type : ObserverType, observer : Observer) : void
+ unsubscribe(observer : Observer) : void
+ notify() : void
<<interface>>
Observer
+ update(code : int) : void
Annotation
- getDocumentState() : State
Comment
Document
+ getState() : State
ExternalData

Figure 3.23: Observer Pattern

The hierarchical nature of the representation of the document, as described in Fig. 3.22, encourages the use of a pattern for traversing the data model in a flexible and customizable way. The **Visitor pattern** provides a mechanism for extending the functionality of the library. Fig. 3.24 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 by means of the client operation. The custom object is then used by the entities of the hierarchy without knowing the behavior of the visitor. In conclusion, the Visitor pattern provides a procedure for operating flexibly on the library data model.

The **Strategy pattern**, instead, allows the definition of a family of algorithms. These algorithms can freely vary independently from clients that use them, according to customizable policies.

Fig. 3.25 shows how this pattern has been applied in the design of the library. The figure illustrates the objects that implement a portion of the text analysis component. The TSLib provides mechanisms for substituting engines and for adapting analysis strategies. Indeed, TSLib exposes functionalities for linguistic analysis by means of language independent entities that dynamically instantiate


SourceLeaf
- accept(visitor : Visitor) : void
SourceComposite
- accept(visitor : Visitor) : void
<<interface>>
Source
+ traverse(visitor : Visitor) : void
Client
<<interface>>
Visitor
+ visit(component : Source) : void
ClientVisitor
+ visit(component : Source) : void

```
visitor.visit(this)
```

```
ClientVisitor visitor = new ClientVisitor();
source.traverse(visitor);
write(visitor.getBuffer());
```

```
buffer.add(component.
getContent())
```

Figure 3.24: Visitor Pattern

the appropriate tools. The analysis objects provide methods that require the content as input and return its analysis as output. The external entities can interact only with the TSLib public component, which creates and uses a special object called **AnalysisContext**. The latter object makes the right association among the language, the linguistic analysis and the engine to be used. For this reason, this pattern represents an important aspect in the design of loosely coupled

APIs.

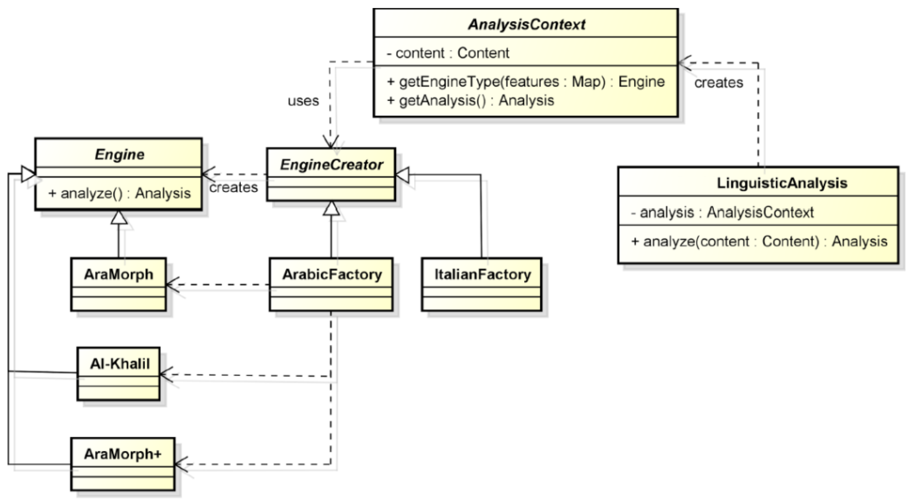
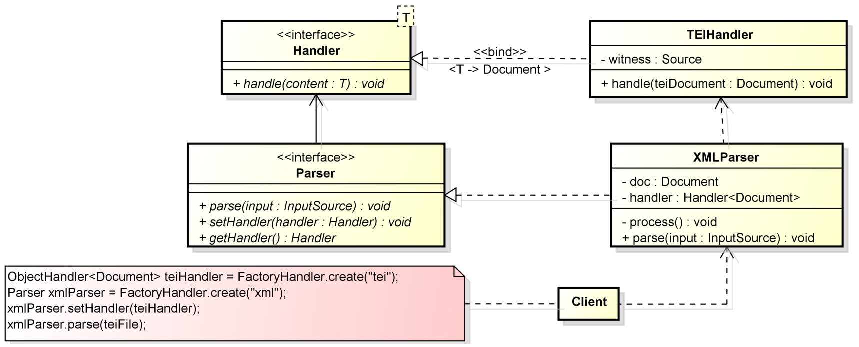


Figure 3.25: Strategy Pattern

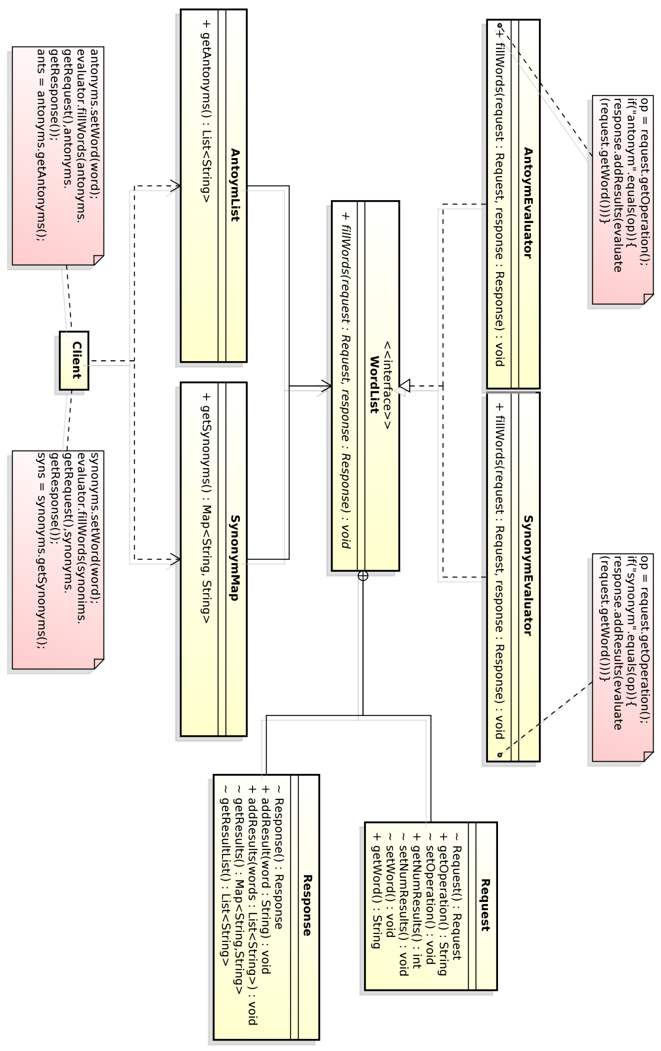
Parser Handler and Response Replay patterns Fig. 3.26 and Fig. 3.27 show two further techniques employed in the design of the library. These are the *Parser/Handler pattern* and the *Response/Replay pattern*.


<<interface>>
Handler
+ handle(content : T) : void
T
<<bind>>
<T -> Document >
TEIHandler
- witness : Source
+ handle(teiDocument : Document) : void
<<interface>>
Parser
+ parse(input : InputSource) : void
+ setHandler(handler : Handler) : void
+ getHandler() : Handler
XMLParser
- doc : Document
- handler : Handler<Document>
- process() : void
+ parse(input : InputSource) : void
Client

```
ObjectHandler<Document> teiHandler = FactoryHandler.create("tei");
Parser xmlParser = FactoryHandler.create("xml");
xmlParser.setHandler(teiHandler);
xmlParser.parse(teiFile);
```

Figure 3.26: Parser Handler Pattern

The first follows the design of the Simple API for XML (SAX) technology. It allows the building of a library able to parse several formats of input files and then process them with the most suitable handler. Therefore, the data format


AntoymList
+ getAntonyms() : List<String>
SynonymMap
+ getSynonyms() : Map<String, String>
Client
<<interface>>
WordList
+ fillWords(request : Request, response : Response) : void
AntoymEvaluator
+ fillWords(request : Request, response : Response) : void
SynonymEvaluator
+ fillWords(request : Request, response : Response) : void
Response
~ Response() : Response
+ addResult(word : String) : void
+ addResults(words : List<String>) : void
~ getResults() : Map<String,String>
~ getResultList() : List<String>
Request
~ Request() : Request
+ getOperation() : String
~ setOperation() : void
+ getNumResults() : int
~ setNumResults() : void
~ setWord() : void
+ getWord() : String

```
antonyms.setWord(word);
evaluator.fillWords(antonyms.
getRequest(),antonyms.
getResponse());
ants = antonyms.getAntonyms();
```

```
synonyms.setWord(word);
evaluator.fillWords(synonims.
getRequest(),synonyms.
getResponse());
syns = synonyms.getSynonyms();
```

```
op = request.getOperation();
if("antonym".equals(op)){
response.addResults(evaluate
(request.getWord()));}
```

```
op = request.getOperation();
if("synonym".equals(op)){
response.addResults(evaluate
(request.getWord()));}
```

Figure 3.27: Request Response Pattern

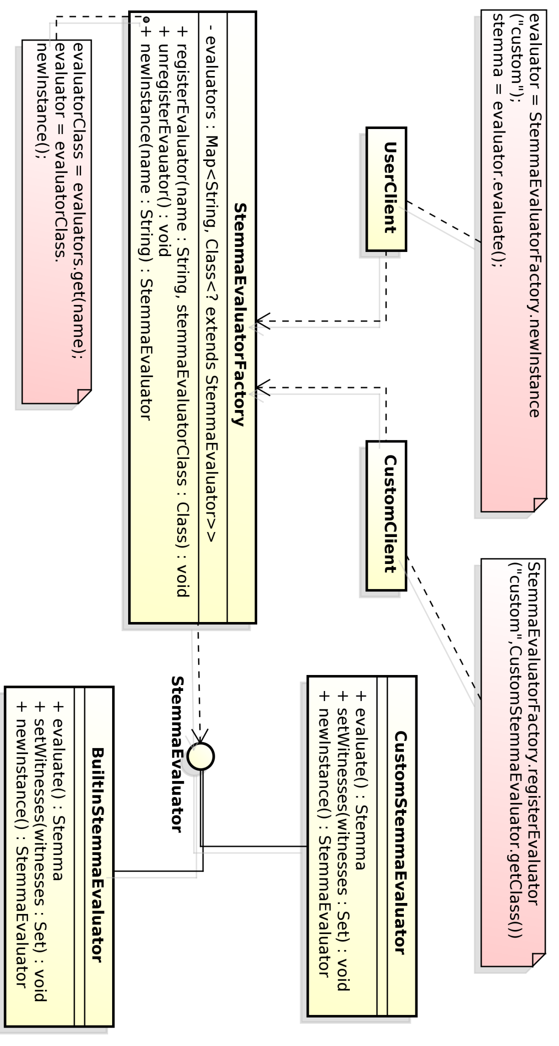
adopted to serialize or persist the textual information does not affect the representation of the textual entities within the library. Input objects, among others, can be in Text Encoding Initiative (TEI), JavaScript Object Notation (JSON), Comma Separated Values (CSV), or even in document (DOC) or Portable Document Format (PDF) (see chapter 4). The parser reads data from an input source in order to produce an object representation of a specified type. By attaching a handler to a parser, the handler will receive a stream of objects produced by the parser. In other words, a parser is created for the format in which the input file is supplied and then a suitable processing tool is created and attached to the parser. The second pattern (Fig. 3.27), instead, is a technique which allows the library to extend its capabilities and to evolve easier without breaking backward compatibility. The pattern solves the issue of how to enhance data exchange with additional parameters.

MVC and DAO patterns Data has to be stored somewhere for persistence purposes, but the library must be agnostic with regards to persistence technologies and persistence data formats. Consequently, an ad hoc module deals with storing data in some kind of database. It is important to note that whereas the entities of the library model represent the actual information stored, the model objects interact with an abstract persistence layer instead of the real database system. This can be achieved by including a Data Access Object (DAO) in charge of handling communication from and to the database. Consequently, the communication between the database and the library is transparent to the core components of the library. Moreover, the DAO component will create all the necessary functionalities to read, store, and modify data in the database. This functionality is known as CRUD (Create, Read, Update, and Delete).

In conclusion, The Textual Scholarship Library also benefits from the advantages of the Model View Controller (MVC) pattern, which provides important features, such as full control between the presentation, the model, and the storage of the library itself. This separation among appearance (view), data (model) and action (controller) is used widely in Object Oriented applications. Hence, the overall architecture of the library (graphical user interface combined with the core library API) ensures separation and decoupling among: (a) the representation of internal data status, (b) rendering, (c) system interaction, (d) user scenarios, and (e) content management. The MVC design pattern, therefore, plays a central role in designing the library components.

3.5.3 Reusability and extensibility

the TSLib capability may need to evolve over time as scholars can meet new requirements. Therefore, it is crucial to design and implement mechanisms that guarantee component reusability and extensibility. This can be done with the use of plug-and-play mechanisms [235]. Indeed, plug-ins are nowadays a common way to achieve flexibility in complex systems. Fig. 3.28 shows the mechanism used to implement new components.


StemmaEvaluatorFactory
- evaluators : Map<String, Class<? extends StemmaEvaluator>>
+ registerEvaluator(name : String, stemmaEvaluatorClass : Class) : void
+ unregisterEvauator() : void
+ newInstance(name : String) : StemmaEvaluator
UserClient
CustomClient
StemmaEvaluator
BuiltInStemmaEvaluator
+ evaluate() : Stemma
+ setWitnesses(witnesses : Set) : void
+ newInstance() : StemmaEvaluator
CustomStemmaEvaluator
+ evaluate() : Stemma
+ setWitnesses(witnesses : Set) : void
+ newInstance() : StemmaEvaluator

```
evaluatorClass = evaluators.get(name);
evaluator = evaluatorClass.
newInstance();
```

```
evaluator = StemmaEvaluatorFactory.newInstance
("custom");
stemma = evaluator.evaluate();
```

```
StemmaEvaluatorFactory.registerEvaluator
("custom",CustomStemmaEvaluator.getClass())
```

Figure 3.28: TSLib Extension Capability

The extension mechanism is based on a factory component which is in charge of handling and organizing the different implementations of a specific service. In particular, the figure shows an example taken from the **stemmaEvaluator** component, where, under the effect of an input, a set of related sources returns the hypothetical *phylogenetic tree* of the text. As argued in chapter 2, the algorithms for this kind of operation are still in the process of being defined by literary computing researchers. Therefore, it is important for the TSLib to have the flexibility to add and to change the implementation for this functionality. In conclusion, the mechanism handles the object-binding at run-time, choosing the actual type of interface. For instance, the actual object of the **StemmaEvaluator** Interface can be known only at run time, when the factory class can instantiate the object by using a dynamic repository and advanced programming mechanisms like the *reflection.* As a matter of fact, the factory object has a registration mechanism to plug and play new implementation components.

3.6 Developing technologies

The prototype of the library makes use of Java programming language. The fundamental artifacts are organized in several Java packages which map the core modules. These packages form a coherent library of data structures and algorithms specifically designed for literary purposes. The prototype has been adopted in a few real applications in the context of funded research projects. Besides this library, other technologies for the development of complex textual processing systems have been taken into account. Such technologies mainly involve the Java Enteprise Edition, the NLP technologies, and the XML related library such as jDOM. The overall system, which is a web-based application, has been developed following the Server Faces Framework (JSF2) and the Model View Controller (MVC) architectural pattern (see section 3.5.2). TEI-Compliant encoding documents are stored in an eXist-db (XML oriented database) and the platform is synchronized with it for the data management. Therefore, the *view tier* is the Web, the *business logic tier* is made up of object defined by library entities, and finally, the *data/integration tier* is achieved by the XML native database. The integrated system is a collaborative multi-layered application and it handles the presentation logic by making use of two complementary Java enterprise technologies: a) *Facelets* and b) *Primefaces.* The first one is a component-oriented technology for Web templating; its benefits are represented by an efficient writing code and an effective software reusing. The second one is a rich and friendly Ajax taglib that allows the development of a flexible user interface (GUI).

Chapter 4

Case Studies

This chapter provides an insight into some issues concerning textual scholarship. A range of effective and practical fieldworks guide the design and the methodological work that has been explained in the previous chapters (see chapter 3) [214]. Consequently, as argued in other sections, the development of Textual Scholarship Library (TSLib) offers a double approach based on bottom-up and top-down methods. Modeling scholar tools by means of these two converging directions allows designers to generalize, extend and refactor the overall architecture as new requirements and common issues emerge. Additionally, when necessary, Agile methods and Application Programming Interface (API) design track the way for arranging new needs. The general architecture, introduced in chapter 3 which encompasses content acquisition, text processing, and data exploitation, has been tested on a number of case studies:

- Source acquisition and text encoding;
- Textual indexing;
- Text alignment;
- Variant reading annotations and multi-level analysis.

It is worth noting that text processing has been performed by integrating open source tools such as Lucene, Tika, Tesseract, eXist-db in to the TSLib experiments (some of them have been introduced in chapter 2). Well-known procedures and measures like Euclidean cosine or algorithms such as the Needleman-Wunsch was adopted throughout the case studies.

4.1 Source acquisition and text encoding

The fundamental objective for scholars who study textual materials with computational methods is to create digital representations of these resources in a formal and suitable format (i.e. machine-readable and machine-actionable). The digital representation of documents offers the possibility to use computing in order to manage and analyze textual data. For example, scholars can perform queries against a corpus as well as extract statistical information from it. Thus, literary computing applications start with the availability of (a) digital images of primary sources, such as a collection of manuscript scans or (b) the electronic transcriptions of original documents. Sources can also be 3D models of the object, a midi or a wave digital document, a music sheet, etc.

With regard to the scope of this work, a few examples of such sources could be

- the image scan of a critical edition;
- the image of a manuscript of a modern author;
- the image scan of a manuscript or codex (Fig. 4.1);
- an unstructured (or bad formatted) electronic document (Fig. 4.2).

The sources, which the software deals with, are complex textual objects. Generally, these objects have non-Latin alphabet with a large amount of phenomena to be annotated at different levels of analysis. Examples of this complexity are:

- multiple author's interventions;
- state of the sources;
- glyphs, typesetting and graphics recognition;
- unusual document structure and layout analysis;
- handwritten character recognition;
- big data for meta-data extraction;
- non-optimal scans of document images.

From the above mentioned issues it derives that digital acquisition of the original sources and effective encoding of digital texts are often the main issues to be overcome in textual scholarship.

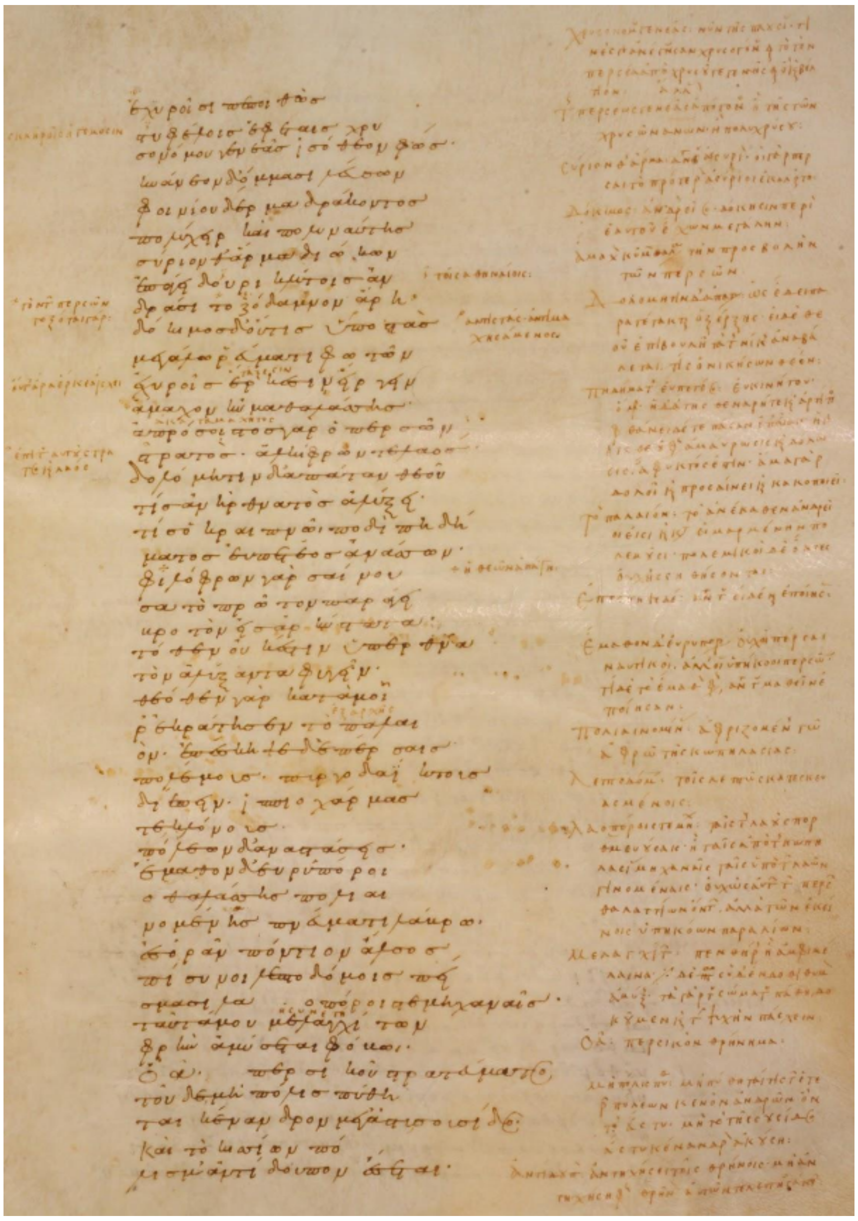
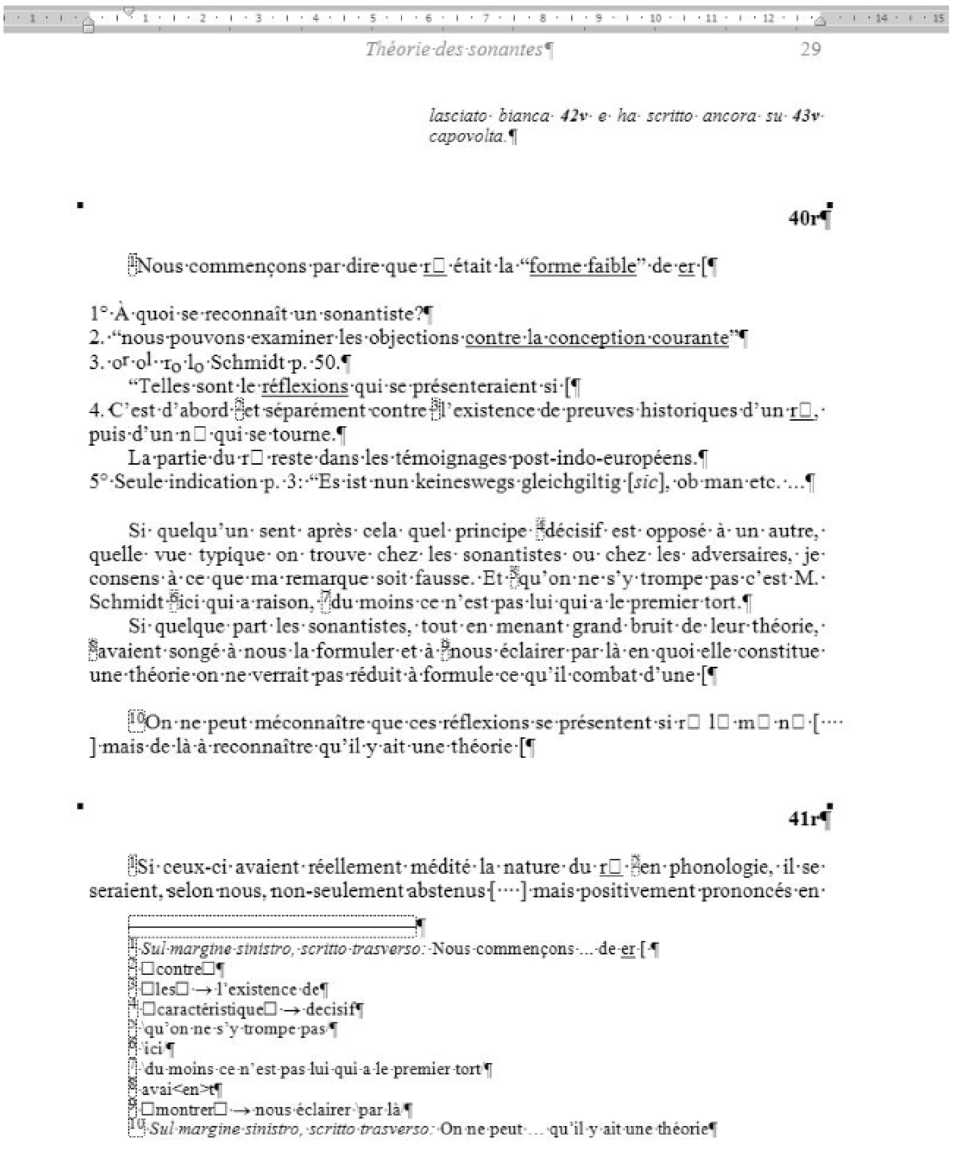

[illegible]

Figure 4.1: Example of a manuscript written in Greek language



Théorie des sonantes 29

*lasciato bianca **42v** e ha scritto ancora su **43v** capovolta.*

40r

[1]Nous commençons par dire que r̥ était la "forme faible" de er [

1° À quoi se reconnaît un sonantiste?
2. "nous pouvons examiner les objections contre la conception courante"
3. o^r o^l r$_o$ l$_o$ Schmidt p. 50.
"Telles sont le réflexions qui se présenteraient si [
4. C'est d'abord [2]et séparément contre [3]l'existence de preuves historiques d'un r̥, puis d'un n̥ qui se tourne.
La partie du r̥ reste dans les témoignages post-indo-européens.
5° Seule indication p. 3: "Es ist nun keineswegs gleichgiltig [*sic*], ob man etc. …

Si quelqu'un sent après cela quel principe [4]décisif est opposé à un autre, quelle vue typique on trouve chez les sonantistes ou chez les adversaires, je consens à ce que ma remarque soit fausse. Et [5]qu'on ne s'y trompe pas c'est M. Schmidt [6]ici qui a raison, [7]du moins ce n'est pas lui qui a le premier tort.
Si quelque part les sonantistes, tout en menant grand bruit de leur théorie, [8]avaient songé à nous la formuler et à [9]nous éclairer par là en quoi elle constitue une théorie on ne verrait pas réduit à formule ce qu'il combat d'une [

[10]On ne peut méconnaître que ces réflexions se présentent si r̥ l̥ m̥ n̥ [···] mais de là à reconnaître qu'il y ait une théorie [

41r

[1]Si ceux-ci avaient réellement médité la nature du r̥ [2]en phonologie, il se seraient, selon nous, non-seulement abstenus [···] mais positivement prononcés en

[1] *Sul margine sinistro, scritto trasverso:* Nous commençons … de er [
[2] ⌜contre⌝
[3] ⌜les⌝ → l'existence de
[4] ⌜caractéristique⌝ → decisif
[5] \qu'on ne s'y trompe pas/
[6] \ici/
[7] \du moins ce n'est pas lui qui a le premier tort/
[8] avai<en>t
[9] ⌜montrer⌝ → nous éclairer \par là/
[10] *Sul margine sinistro, scritto trasverso:* On ne peut … qu'il y ait une théorie

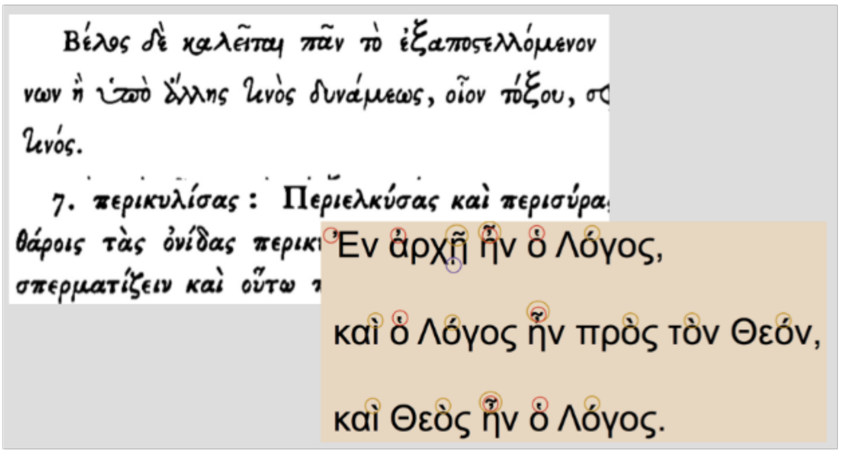
Figure 4.2: Example of a word processing electronic file

4.1.1 Text acquisition

As explained in the previous chapters, the automatic reading of printed text can be done by Optical Character Recognition (OCR), which converts digital images in machine-readable text. On the contrary, the extraction of information can be performed through document manipulation tools. These can be run from unstructured data in order to obtain well-formed and structured resources (see section 2.6).

Standard procedures, developed to perform OCR on general purpose collections of books, yields poor outcomes for critical editions [236, 237, 238]. In fact, this kind of resources contains critical aspects as polytonic Greek and multi-lingual critical apparatuses.

Thus, specific OCR procedures need to be developed and then applied to the scanned books. Thereafter, the corrected digital text must be remapped on the original page images. OCR systems applied to printed editions that contains texts in Greek, Latin or Arabic, require sophisticated algorithms and methodologies, both in pre-processing and post-processing phases, such as the alignment of multiple OCR outputs for improving the accuracy [22, 239]. As discussed above, text recognition and information extraction from critical editions is not a trivial operation. Besides having glyphs with complex patterns (Fig. 4.3), page layout is usually divided into several text flows with different graphical conventions (text, apparatus, notes). Due to the large amount of computation on a lot of data, the production and manipulation of resources, require high performance computing environments and process parallelization on a grid of supercomputers [240].


Βέλος δὲ καλεῖται πᾶν τὸ ἐξαποστελλόμενον
νων ἢ ὑπὸ ἄλλης τινὸς δυνάμεως, οἷον τόξου, σ
τινός.
7. περικυλίσας: Περιελκύσας καὶ περισύρα
θάροις τὰς ὀνίδας περικ
σπερματίζειν καὶ οὕτω

Ἐν ἀρχῇ ἦν ὁ Λόγος,
καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν,
καὶ Θεὸς ἦν ὁ Λόγος.

Figure 4.3: Example of Greek glyphs - Courtesy of prof. Bruce Robertson

The challenge of the case study that has been introduced here consists in the acquisition of printed critical editions in historical languages such as Greek and Latin, but also Arabic. The OCR engines used during the textual scholarship experiments were open source packages, namely Gamera, Tesseract and Ocropus.

As pointed out by [239, 238] during their experiments the recognition of a page could require about two minutes of processing per CPU core. Hence, it is natural to parallelize the process of digitization. Other experiments, moreover, have shown it is possible to significantly improve the accuracy of the results by applying alignment techniques on recognized texts [22]. In this framework, parallelization has a double benefit. Indeed it allows to: (a) decrease the time required for massive acquisition of texts (from the order of years to the order of months); (b) implement strategies to increase the accuracy. In turn, this can be done by:

- choosing the optimal parameters for image enhancement in pre-processing,
- choosing different training-sets (classifiers) for OCR-engines in processing,
- aligning and correcting results in post-processing through linguistic tools such as spell-checkers.

This case study has exploited the HLRS Environment (Fig. 4.4).

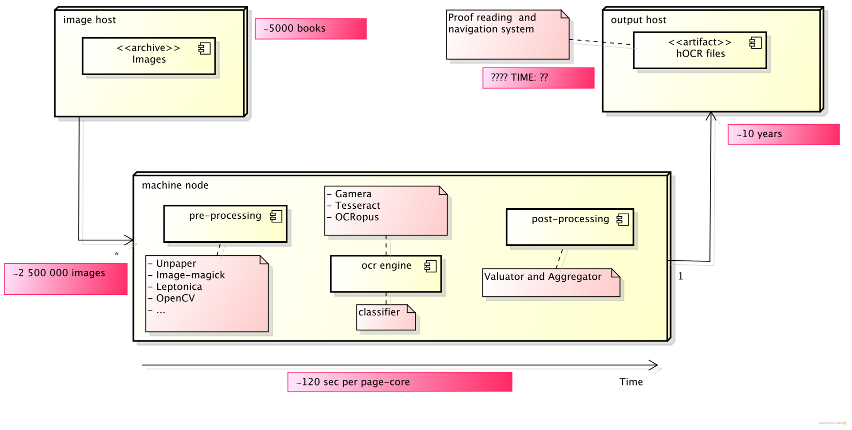


Figure 4.4: OCR experiments conducted on the HLRS Environment

Each page, assigned to a single core, requires on average 2*minutes* for optical character recognition. Each page will be processed by three OCR engines with different classifiers: *Gamera*, *Tesseract* and *Ocropus* (=15*minutes*). 10 additional minutes for page preprocessing must be taken into account, in order to tune the image parameters and select the best training sets. 5*minutes* for each post-processing and text analysis require 5*minutes* for each page. Thus, 30 minutes per page are required for the entire process. The historical corpus is constituted by 5000 books counting 500 pages on average. The experiments have been performed on the Hermit installation based on CRAY XE6 technology consisting of 3552 nodes. Hermit provides *1.045 Pflops* as peak performance and has two sockets per node with 16 cores.

The processor in each node is a *Dual Socket AMD Interlagos* at 2.3*Ghz*. The interconnection between the nodes is made by *High Speed Network Cray Gemini* with *HyperTransport HT3* and it can achieve a rate of *102.4 GB/s.* The compute nodes for running parallel jobs are only available through the Portable Batch System (PBS) and by means of the Application Level Placement Scheduler (ALPS). The workspace file system has a capacity of *2.7 PB* and a IO bandwidth of *150GB/s* (*Lustre parallel system*). The operative system is the Cray Linux Environment (CLE), which is based on SUSE Linux Enterprise Server (SLES). Finally, Message Passing Interface (MPI) has been used for the parallelization. Cray Compiler has been used to make the binary files executable. Thanks to PBS and ALPS, jobs are submitted to run on the grid.

Leptonica has been used in pre-processing, in order to perform orientation fixing, line segmentation, content selection, resolution adjustment, dewarping and despeckling. These operations improve the image readability for character recognition. Whereas Tesseract has been used to perform OCR. Evaluation of the text after OCR has been performed and the number of recognized words (scored highest), pseudo-words (i.e. well formed syllabic sequences), or sequences of random symbols have been weighted following the formula 4.1.

$$G_{score} = \sum_{j=1}^{totchar} \frac{k_j nchar_j}{totchar} \tag{4.1}$$

The developed software module performs dispatching with a Master-Slave topology of the super computer grid. The right hand part of Fig. 4.5 shows the time implied to apply OCR on 511 different images with best parameters. The parallel process involved 2 nodes (7073 sec) and 8 nodes (1005 sec). Time lapses have been tested also with 32 (231 sec), 128 (61 sec), 256 (38 sec) and 512 (21 sec) nodes. Since Slaves communicate only with the Master, the time required increases by

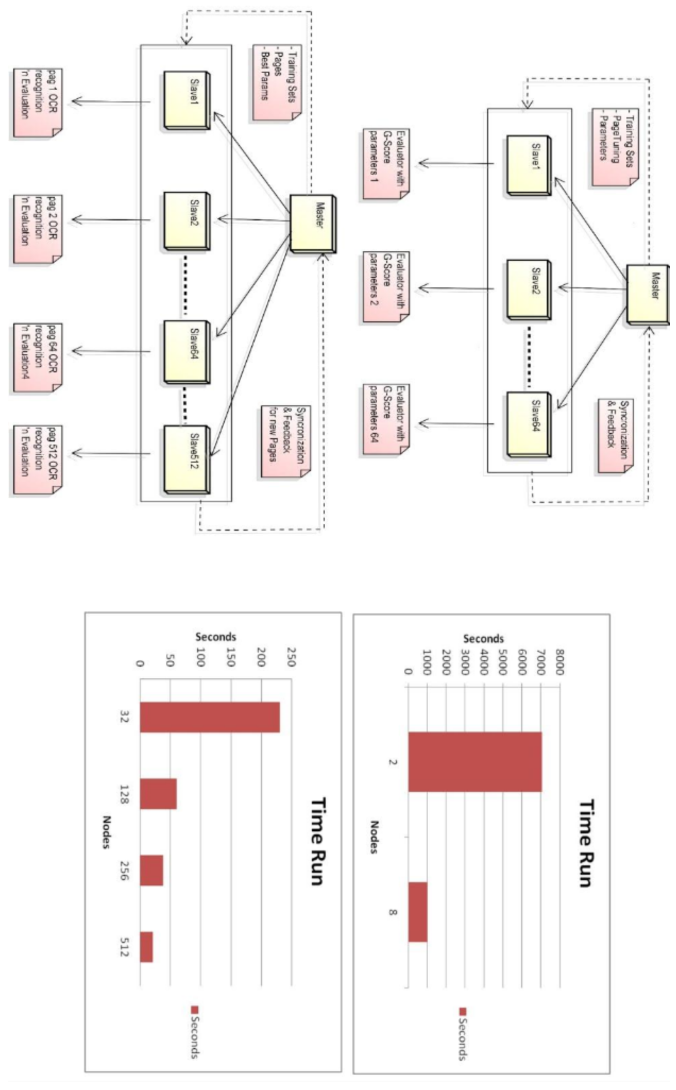

- Training Sets
- Pages
- Best Params
Slave1
Slave2
Slave64
Slave512
Master
pag 1 OCR recognition n Evaluation
pag 2 OCR recognition n Evaluation
pag 64 OCR recognition n Evaluation4
pag 512 OCR recognition n Evaluation
Syncronization & Feedback for new Pages
- Training Sets
- PageTuning
- Parameters
Evaluator with G-Score parameters 1
Evaluator with G-Score parameters 2
Evaluator with G-Score parameters 64
Syncronization & Feedback
Time Run
Seconds
Nodes
32
128
256
512
0
50
100
150
200
250
2
8
1000
2000
3000
4000
5000
6000
7000
8000

Figure 4.5: Parallel OCR experiments evaluation

an inverted-proportion law law related to the nodes used. Recognitions are available both in plain text and in HTML-based Markup for OCR (hOCR) formats. The latter includes elements and properties about the word box coordinates of the recognized words in HyperText Markup Language (HTML) tags.

Two experiments have been performed. The first experiment concerns parameters tuning upon the nodes of the grid. This has been done in order to identify the best combination to improve the accuracy of the recognition. The second experiment concerns the application of OCR with the best parameters on sample pages by a *divide et impera* strategy. In turn, the experiments concern two tasks: (a) improvement of the OCR accuracy; (b) reduction of the time needed to perform the recognition. As announced, the accuracy of OCR engines applied to polytonic Greek can be improved in three phases: in pre-processing, by image adjustment, during the recognition, by a suitable selection of training sets and in post-processing, by the alignment of the output of different OCR engines. Finally, parallelization can speed up all the aforementioned phases. Indeed, the parallelization of the OCR processes on a grid of supercomputers reduces the time for computation and promotes improvement of accuracy by pursuing multiple strategies.

The work has the focus on improving accuracy by handling and manipulating two parameters on image *binarization* process. Two piped functions on a selected page perform contrast normalization of the background and *binarization* using *Sauvola algorithms* [241]. The corpus contains text both in Latin and polytonic Greek character sets. Accordingly, the OCR experiment on the corpus encompasses two different tasks, in order to measure the time saved thanks to the parallelization: (a) the first task ran on a number of nodes less than the total number of pages; (b) the second task ran on a number of nodes that corresponds to the total number of pages.

Fig. 4.6 illustrates how the pre-processing of a page with different parameters is reflected in the expected accuracy.

64 nodes in parallel perform OCR on the same selected page, in order to tune parameters, according to the following rules:

$$IDnorm = \frac{rank_i}{8} \tag{4.2}$$

$$IDbin = rank_i \bmod 8 \tag{4.3}$$

In this way, a vector of different parameters has been created and each node, based on its process identification $rank_i$, determines a combination of parameters:

$$< IDnorm, IDbin > \tag{4.4}$$

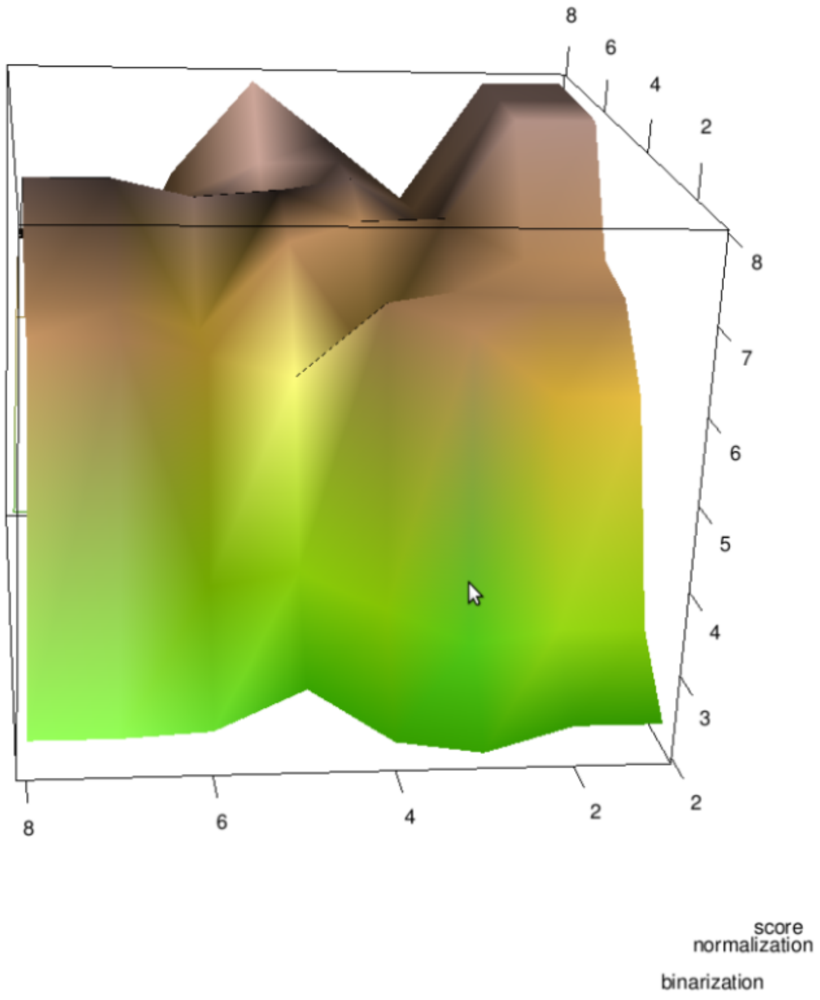


Figure 4.6: Image processing parameters - Courtesy of F. Boschetti

Fig. 4.7 illustrates how to assess the accuracy of large-scale OCR applied to Ancient Greek texts by estimating whether the Greek words are correctly recognized.

The diagram shows a weighted way to evaluate the Greek text recognition. The score is important to compare different OCR outputs obtained with different approaches on the same page images. In the current work, the score is relevant only to order the OCR performances; its absolute value is largely underestimated,

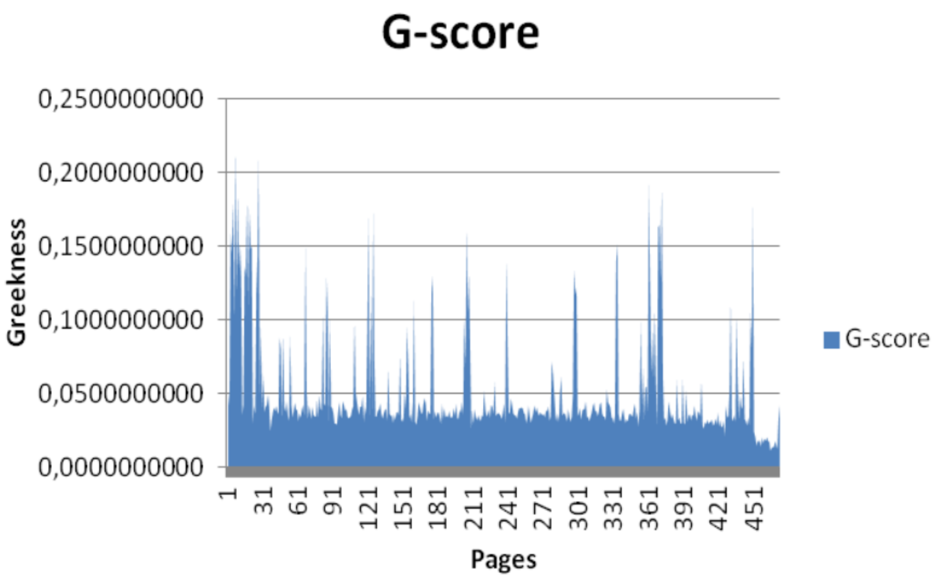


Figure 4.7: Greekness score evaluation

due to the presence of Latin words, which are wrongly considered as errors.

4.1.2 Character and text encoding

Designing applications for literary computing and writing software to process historical text pose challenges on the point of view of character and text encodings [242]. As introduced in section 2.4.1 this encompasses at least two types of problems:

- Character encoding;
- Document encoding.

This section presents a case study within a PRIN project, in which specific principles have been applied in order to handle both legacy and Text Encoding Initiative (TEI)-The eXtensible Markup Language (XML) documents (Fig. 4.8), as described in chapter 3.

The TSLib component for content management aims to make the documents to be processed within a collaborative environment. This component is in charge of analyzing texts represented by formal schemes, through shared and standardized markup languages (see chapter 2).

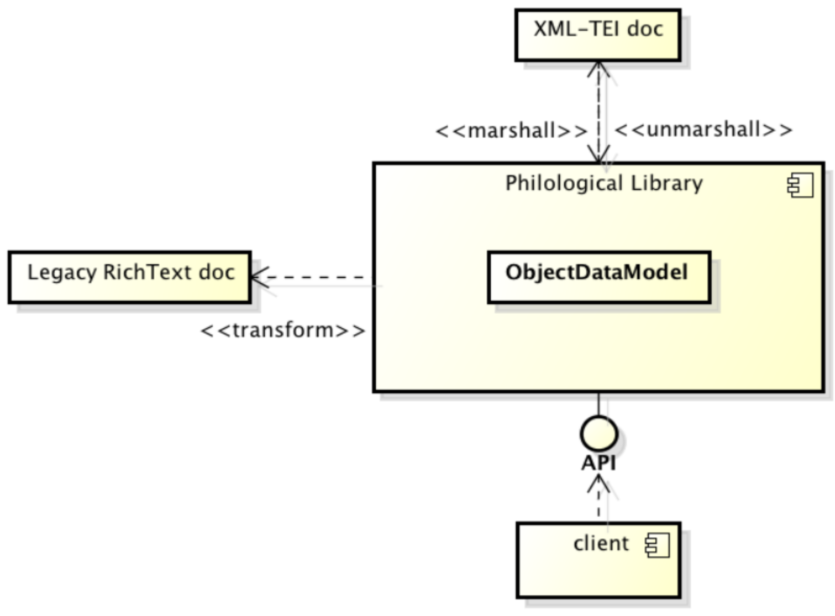


Figure 4.8: Data Model versus External Data Format

Fig. 4.2 shows a page from one of the electronic documents processed by the library, in particular, page 29 of the *Theorie de Sonante* book edited by Marchese. From the picture that reports f. 40r of the manuscript *BGE Ms. fr. 3955/1* it is possible to note the transcription work done by this scholar. The data recovery component must handle the following aspects: a) erroneous interpretation of certain text characters; b) the publisher's notes in the text; c) use of titles to indicate the number and the beginning of the folio; d) use of stylistic conventions such as italic, bold, underline, type of font, etc.; e) footnotes with publisher contributions for the reconstruction of the history of the manuscript;

The component in Fig. 4.8 is functional in order to process the resource and extract the information contained in the document.

The latter needs be processed through the assignment of the binary code as indicated by the Unicode standard coding system.

Problems occur when non-standardized systems are used with an aim to simulate graphically these symbols. In these cases, a systematic correction is necessary in order to retrieve the original intention of the editor (Marchese) and, consequently, of the author (Saussure). In general, all the characters that do not have any correspondence in this coding table display problems.

The data acquisition module recognizes such problems and uses techniques that base their procedures on regular expressions, on statistical functions and on heuristic processes.

In the pre-processing phase the transcode operation is performed and the document is encoded in a well-formed manner. The reading of the text is done initially character by character; this procedure tries to figure out if the associated binary code is valid or if it needs to be verified. At the moment this operation follows rule-based procedures. Nevertheless, the component can be extended by means of statistical mechanisms (see chapter 2). The transcoding module of the Library enlists all the codepoints considered invalid and associates a correct substitute codepoint thanks to heuristics and semi-automatic procedures. The characters associated to an invalid coding are compared to appropriate conversion tables in order to produce the correct substitutions [243]. The character substitution module can be extended through more complex systems based on spellcheckers or on statistics of n-grams co-occurrences. Subsequently, the original file in legacy format triggers a second software component that aims to isolate stylistic and structural information. As an example, in Fig. 4.2, it is possible to identify: (1) the number of the folio (*f. 40r*); (2) the content of the sheet (all the text extracted between two successive sheets); (3) the footnotes (the text of the footnote and the position of the relative associated word); (4) all the document styles. Two software libraries have been employed in the development of the above described component, the first one is able to facilitate the access and the reading of a document (TIKA, see section 2.5.1) and the other is adequate methods for the software programmer APIs to generate and manipulate data in XML (jDOM).

At the end of this process each textual entity of the collection (*Source Class* see chapter 3) is mapped in the library data model (see section 3.4). The library provides a series of methods in order to serialize the digital representation of texts documents in a file which is XML TEI compliant [194] (Fig. 4.8).

Objects that represent the whole document or interrelated documents are initialized through the parsing of the original document and the creation of a new data structure. The latter decouples the orthogonal information conveyed by the XML elements: a) textual structure, b) semantics, c) style, and d) behavior. Noticeably the new data structure can result from the transformations (by eXtensible Stylesheet Language Transformations (XSLT) Document Object Model (DOM) transformations or Simple API for XML (SAX) event driven transformations) managed during the parsing process. In turn, as explained in chapter 3, each textual entity is composed of a **version**, a **granularity**, an **interpretation** and a **position** in the document. Therefore, the parsing process concerns the following aspects:

1. **Textual structure**. A document originally structured paragraph by paragraph for literary analysis can easily be restructured page by page to pass through layout analysis or to be compared with the original page image.

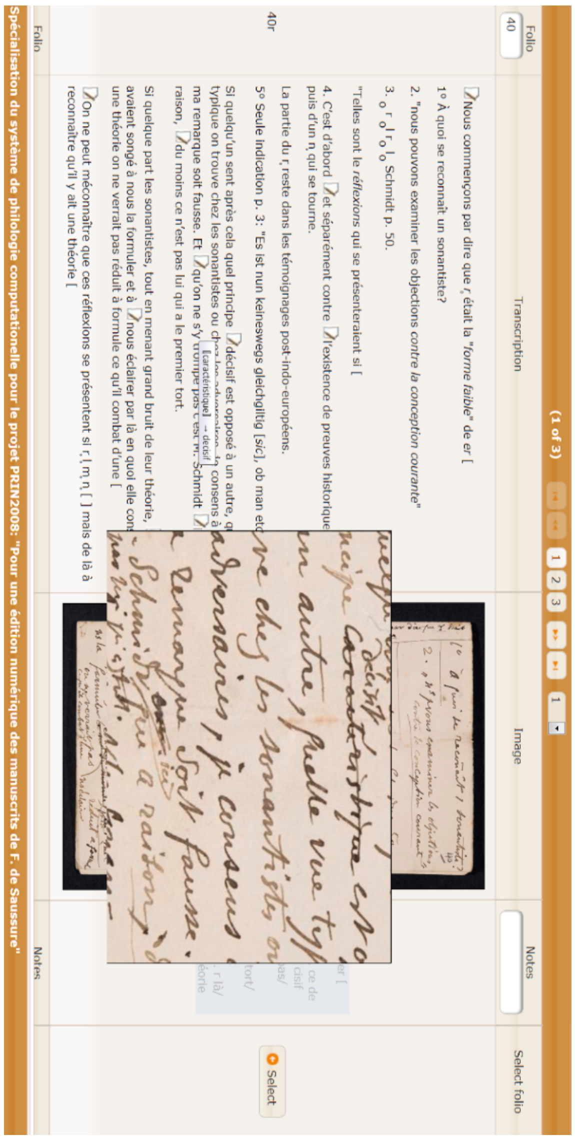
2. **Semantics**. At semantic level, both attributes (such as `@type`) and tag names (such as <p/>) are processed in the same way and linked to the related DOM node.

3. **Style**. The style is managed by separated components, which point at textual positions affected by stylistic features. For instance, the information extracted from the `@style` attribute is used to instantiate the Java objects devoted to manage the rendering information.

4. **Behavior**. Behaviors are handled by object 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 (e.g. the hyphenation rules for the Italian language).

The textual scholarship library, unlike most of current initiatives which focus on the transformation from an XML document structure into another by XSLT, instances a collection of objects which map the representation of the supplied documents (see chapter 3). The aforementioned case study considers only a small subset of TEI elements as basic taglib (see section 4.4). Fig. 4.9 illustrates the composite of widgets rendered on the client through rich standard web technologies (HTML5, CSS3, JQuery, D3.js). The Graphical User Interface (GUI) allows scholars to browse both texts and images. The links in Fig. 4.9 are referred to annotations taken by the author of the manuscripts and to which the editor refers in the critical apparatus.

The acquisition component reads the input document and dynamically generates the XML schema (see section 3.5.2). Factories (see section 3.5.2) instantiate the actual object implementing or extending the interfaces (or abstract classes)



40r

Nous commençons par dire que r̥ était la "*forme faible*" de *er* [

1° À quoi se reconnaît un sonantiste?

2. "nous pouvons examiner les objections *contre la conception courante*"

3. r̥ l̥ r̥ l̥ Schmidt p. 50.

"Telles sont le *réflexions* qui se présenteraient si [

4. C'est d'abord et séparément contre l'existence de preuves historique
puis d'un n̥ qui se tourne.

La partie du r̥ reste dans les témoignages post-indo-européens.

5° Seule indication p. 3: "Es ist nun keineswegs gleichgiltig [*sic*], ob man etc

Si quelqu'un sent après cela quel principe décisif est opposé à un autre, q
typique on trouve chez les sonantistes ou ch [caractéristique] → decisif consens à
ma remarque soit fausse. Et qu'on ne s'y trompe pas c'est M. Schmidt
raison, du moins ce n'est pas lui qui a le premier tort.

Si quelque part les sonantistes, tout en menant grand bruit de leur théorie,
avaient songé à nous la formuler et à nous éclairer par là en quoi elle con
une théorie on ne verrait pas réduit à formule ce qu'il combat d'une [

On ne peut méconnaître que ces réflexions se présentent si r̥ l̥ m̥ n̥ [ ] mais de là à
reconnaître qu'il y ait une théorie [

Spécialisation du système de philologie computationelle pour le projet PRIN2008: "Pour une édition numérique des manuscrits de F. de Saussure"

Figure 4.9: The GUI the text-image framework

that handle the correct textual-phenomena. For instance, the application client that uses the library invokes the building method of a builder class. The resulting document object is a concretization of an abstract class representing the current structure of the input resource, as illustrated in the Java statement:

```
Source teiDocument = BuilderFactory.buildDocument(
new File(‘‘features.properties’’),
new File(prop.get(‘‘sourceDocument’’));
```

The builder object needs two input files: a) a property file containing the suitable configuration for the instantiations of the concrete objects; b) the XML-TEI file to parse. The state of the internal representation of the document is composite-component based and complies with the data model introduced in chapter 3, section 3.4. This way, each single element is handled by the **Source Class**, which is the component entity of the pattern. It represents each node of the hierarchical structure. Moreover, the information conveyed from the TEI file is distributed among the appropriate Java objects that handle the four levels described above. The leaves of the hierarchical structure are instances of the Text class. The methods in such a structure offer the possibility to manipulate the content and the structure of the resources. As a result, the actual implementation of the of the library content exposes methods that parse the XML file and creates 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 have been used for indexing the textual data.

As mentioned in chapter 3, different collections of texts can provide or ignore some extratextual information (such as line or page number), or they can dispose texts in different ways (e.g. lines can be grouped or not inside <lg>...</lg> elements of the schema, etc.). For this reason, the XSD schema is generated a posteriori from the actual representation of texts. Studying the schemes, XSLT transformations are created, in order to deal only with relevant information and canonical formats processed by the suitable Components.

Finally, in spite of the data structure being an object-oriented representation of the entities in the real domain of the digital document, the storage paradigm is customizable through the adoption of integration mechanisms and data access solutions [200].

This case study shows that standard file formats to encode and exchange textual data have to be handled and abstracted by Application Programming Interfaces. In this way TSLib components allow scholars to read, write, and transform file formats. In addition, API makes it possible to change the format of a file safely.

4.2 Indexing

Once acquired, the correctly codified and stored textual resources are available for further analysis and processing.

The library for textual scholarship deals with several important matters (see chapter 3 section 3.2). Among them, indexing is one of the fundamental functionality for literary computing needs [244]. Textual scholarship applications require indexing features for different kinds of textual phenomena such as domain terminology or lemmatization. It derives that the search component of the library must handle indexes (and concordances) for each of the information that scholars deem significant to study. To this purpose, for instance, multi-languages component indexes have been developed [243].

Index is generally an auxiliary structure able to ensure efficient access to information, following to an external request [245]. Therefore, the philological library must handle automatic and dynamic indexing of the textual data. In order to do so,, the library includes a component that deals with organizing and retrieving relevant textual-phenomena from document collections. At the same time, textual scholar applications must provide accurate results in a short time, as pursued by information retrieval specialists through the study of new methods and techniques. In order to achieve that, literary computing needs software modules able to create indexes for storing and manipulating data, in order to effectively support scholar activities [246]. Through this process indexes allow scholars to identify and retrieve textual context containing the terms they are searching for. Moreover, index components allow to group the content of a text in lists which are usually ordered alphabetically by word-forms, word-lemmas, word-frequencies, and so on. This method is one of the cornerstone tools in the textual inquiries and it is known as "index and concordances" [246]. Noticeably, the textual scholarship library has two distinct entities for indexing as the following list points out:

- back-end index: it represents the internal "data structure" of the stored information (it means how raw data are organized inside the computer persistence unit);

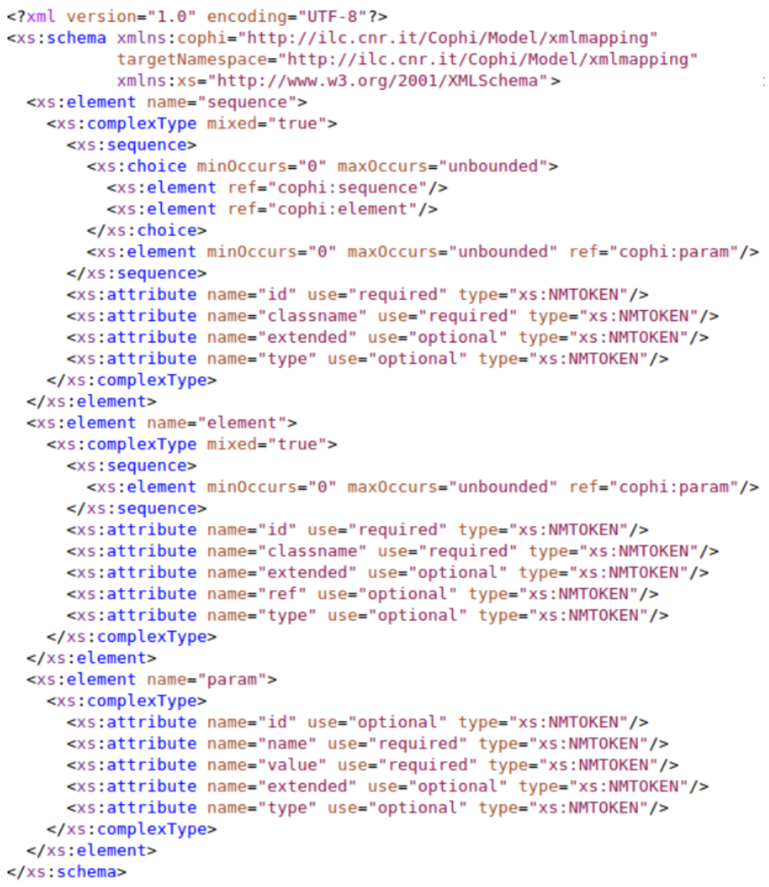
- front-end index: it indicates a uniform and ordered list of key-terms (which may be the chapter headings, paragraphs, word-forms of text or the lemmas, extracted through text processing or linguistic analysis, etc).

Currently, the component *adapts* the Lucene library (see chapter 2 section 2.5) thanks to specific Adapter classes (see chapter 3 section 3.5.2). The indexing component, developed in the context of the aforementioned case study, allows scholars to identify and locate relevant parallel contexts having custom granularity and inter-linking features (see chapter 3 section 3.4). This means that the data model manages different objects at different levels of granularity which, often resulting in overlapping structures (see chapter 2 section 2.6.1). In general, the component deals with four kinds of typed-sources (see chapter 3 section 3.4: token, lines, sentence, and textual fragment. Fig. 4.10 shows the XML schema implemented for marshalling process. The latter follows the TSLib data model according to the composite pattern introduced in the chapter 3 section 3.4 and 3.5.2. The model allows data to be both represented at different levels of granularity and also, to be linked together.

The indexing process developed within the case study aims at building a systematic index which allows to perform search operations. Currently, the process implements several piped activities: 1. tokenization, 2. filtering/normalization, and 3. segmentation. It is possible to see these tasks as a workflow where data input derives from the previous job and data output is formatted and structured so that it can be used by further processes [247].

Tokenization A token is a sequence of alphanumeric codes detected from the electronic text. It is an independent and uniform elaboration unit. Tokenization, therefore, is an activity devoted to the identification of tokens from a text stream. In most Western languages a token corresponds to a graphic form preceded and followed by a blank space or a punctuation character. Tokenization is a source of problems for many textual scholarship phenomena such as the correct handling of punctuation (e.g. in presence of abbreviations and acronyms), as well as when multi-words need to be taken into account. In languages such as Suhaili or German and Arabic, which present agglutinating features, a whole sentence can be expressed by a single sequence of characters. In these cases a tokenization algorithm has to include a segmentation phase in order to recognize the basic units.

Filtering/normalization Information is cleaned up and enhanced through the filtering/normalization phase: each token is analyzed and submitted for further processing. The actions usually performed to accomplish this task are (i) management of stop words, (ii) identification of stem, (iii) use of thesauri and spell-



```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:cophi="http://ilc.cnr.it/Cophi/Model/xmlmapping"
           targetNamespace="http://ilc.cnr.it/Cophi/Model/xmlmapping"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="sequence">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="cophi:sequence"/>
          <xs:element ref="cophi:element"/>
        </xs:choice>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="cophi:param"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="classname" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="extended" use="optional" type="xs:NMTOKEN"/>
      <xs:attribute name="type" use="optional" type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="element">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="cophi:param"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="classname" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="extended" use="optional" type="xs:NMTOKEN"/>
      <xs:attribute name="ref" use="optional" type="xs:NMTOKEN"/>
      <xs:attribute name="type" use="optional" type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="param">
    <xs:complexType>
      <xs:attribute name="id" use="optional" type="xs:NMTOKEN"/>
      <xs:attribute name="name" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="value" use="required" type="xs:NMTOKEN"/>
      <xs:attribute name="extended" use="optional" type="xs:NMTOKEN"/>
      <xs:attribute name="type" use="optional" type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 4.10: XML schema implementing the TSLib data model

checking, and (iv) association of a score to the words (weighting). Scholars often know what to look for, but they ignore how to describe it formally [236]. Hence, index components provide several techniques as stemming and thesauri in order to face the aforementioned issue. Both the stemming techniques, which reduce textual units to common sequences of characters and the use of thesauri, which allow search engines to replace similar terms with canonical ones [248], tend to increase, the amount of the information conveyed by the original token. Doing so, textual resources grow in relevance and can also be retrieved by queries containing similar key-words. In addition, a token derived from a word spelled incorrectly may undergo correction before being stored in the index (under certain conditions as in the case of errors, they are also significant information for the philologists). However, the last action is justified by the user's unawareness of a possible orthographic mistake inside the text (errors from the original text or errors resulting from digital acquisition).

All that considered, it may be necessary to assign a score to evaluate the relevance of textual resources. The textual scholarship library, as introduced in chapter 2 section 2.6 provides a "weight/score" words assignment for partial matching functionalities as well as boolean retrieval features. Partial matching functionality concerns methodologies based on statistical and similarity operations, while boolean functionality refers to exact matching algorithms and it deals with techniques and methodologies where a search result is expressed by boolean operations (i.e. OR, AND, NOT).

The words in a text have different importance and the library component for indexing identifies the differences among words. That means that the term frequency inside a document or within an entire collection can be evaluated and attained as a first weight criteria. Along with full text indexing, the library has developed functionality in order to upgrade raw textual data with extra information (see section 4.4). Typical evaluation parameters used for indexing are search speed, exhaustiveness, specificity, precision and recall. For further details about these topics, please refer to the wide literature (e.g. [146, 249]).

Segmentation Text resources have been segmented into uniform fragments in order to create parallel textual units ruled by consistency [44]. In particular, segmentation and successive connection among parallel textual segments are based on multiple aspects, such as semantic or/and linguistic observations, on which scholars operate thanks to the annotations recorded in the system (see section 4.4). The granularity of the division is such to obtain benefits for the automatic analysis of resources as well as for individual scholarly analysis. Each chunk has a universal identification number, furthermore, related textual chunks also have a universal identifier as the formula 4.5 shows and as illustrated in chapter 3 section 3.4The latter identifier determines the relation between textual chunks.

$$Idpair_n = f(< idchunk_i, idchunk_j >) \tag{4.5}$$

Unique identifiers assigned to each textual entity along with the division of a text into homogeneous segments facilitates indexing process. This because textual elements have a well-defined indexing unit and significant contexts are available in the retrieved result sets.

The list below shows the definition of the back-end index attributes, which are populated by the index component of the designed library component. The ConLL[1] data format, which is a common representation of data largely adopted in the natural language processing field, has inspired the schema of the aforementioned index. The index has a number of attributes that reflect the following points:

- the **Token** field represents the processing unit extracted from the text, where all the subsequent steps of the analysis are grounded;
- the **Fragment** field consists of the identification number (ID) of the fragment whose the token belongs to;
- the **Offset** field expresses the position of the token within the fragment. Thanks to this data it is possible to take into account the proximity relations;
- the **Lang** field refers to the language or alphabet of the token;
- the **Status** field defines additional information: for example, the token could be part of a polyrematic term or express agglutinative phenomena. Terms with graphic variants or tokens belonging to a hyphenation term can be managed through the status field;
- the **Normalization** field provides the possibility of transforming and/or harmonizing the tokens, e.g., yielding the word in uppercase form (as in Greek) or eliminating the sign
- The **Extension** field is left free for any future customization, such as the management of the variant readings;
- The fields **POS**, **Lemma**, and **Root** specify the morpho-syntactic information derived from of linguistic analysis.

A rich index makes it possible to carry out techniques for better document retrieval as an accurate result of the query. Consequently, the index process instances data structures as inverted index [249], this consists of a set of records

[1]for further detail please visit http://ilk.uvt.nl/conll/

containing the word wanted and a sequence of pointers directed towards information related to it, for each significant phenomenon. These kinds of structures are constituted by a list of two-dimensional vectors <k,d> where *k* stands for the key-term and *d* stands for a list of references to the text. The index can store statistical parameters, such as the term frequency of a single document or of the whole collection/corpus. One of the relevant outcomes of this case study is the possibility of performing combined advanced searches, i.e. to find text contexts within the scope of a language while applying restrictions on the parallel one. This special approach provides a method for studying complementarity or links among the texts.

In conclusion, scholars are able to perform queries based on a rich index which includes several data such as: a) linguistic difference between a term and another; b) the position in which textual chunks appear; c) the frequency with which a term appears in the document; d) the status of the search term.

4.3 Alignment

As mentioned in chapter 2, Computational philology requires procedures able to align different kind of textual entities at different levels of granularity. For instance, comparing the witnesses of a text for discovering their differences and supporting the editing of a critical apparatus [6, 35, 250]. Moreover, this kind of entities can differ both for their digital structure and for their inherent nature. For example, as explained in [239], scholars often need to align texts at character granularity in order to evaluate the accuracy of the OCR tools (i.e. the alignment of automatic outcomes with the ground-truth), furthermore, applications like CollateX [80] (see section 2.3.4) need to align different texts at word granularity in order to compare variant readings (Fig. 4.11), as well as cross-lingual studies align texts in different languages in order to investigate different aspects of a literary tradition (Fig. 4.12).

From a computer science point of view, the alignment process uses data structures as n-grams, tries, and suffix trees to speed-up the computation and to reduce space usage (see chapter 3). In addition, fuzzy matching techniques, as well as machine translation methods have common tools adopted in textual alignment applications [45, 251].

The present case study aims at aligning different documents in different languages. The module attempts to segment related texts by means of textual pro-

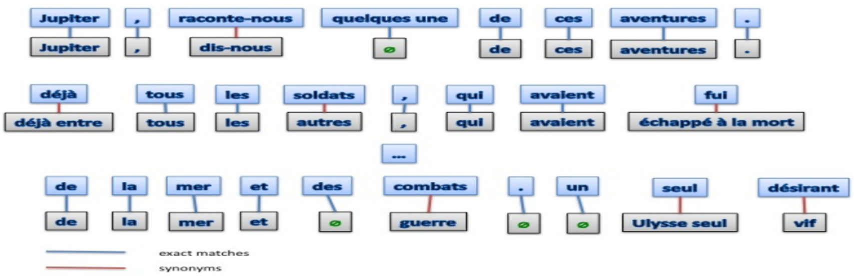
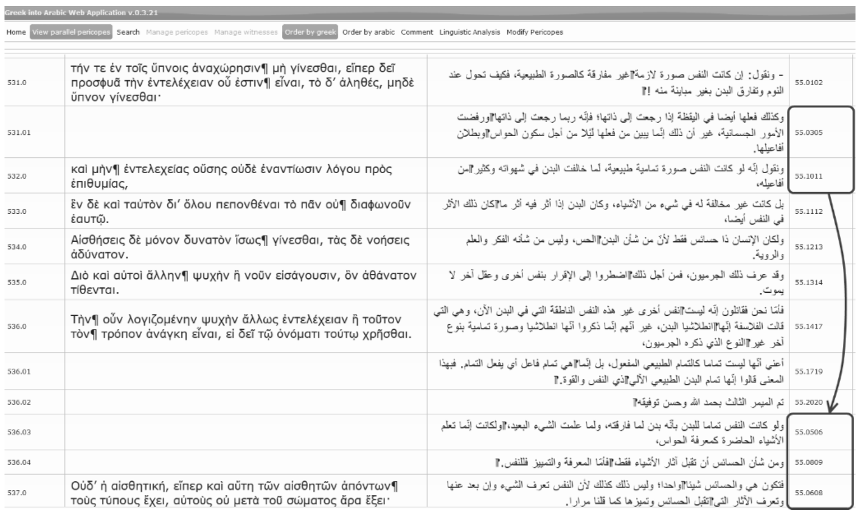


Figure 4.11: Output of the aligner on Odyssey French translations


Greek into Arabic Web Application v.0.3.21

Home | View parallel pericopes | Search | Manage pericopes | Manage witnesses | Order by greek | Order by arabic | Comment | Linguistic Analysis | Modify Pericopes

531.0	τήν τε ἐν τοῖς ὕπνοις ἀναχώρησιν¶ μὴ γίνεσθαι, εἴπερ δεῖ προσφυᾶ τὴν ἐντελέχειαν οὗ ἐστιν¶ εἶναι, τὸ δ’ ἀληθές, μηδὲ ὕπνον γίνεσθαι·	- ونقول: إن كانت النفس صورة لازمة⁋غير مفارقة كالصورة الطبيعية، فكيف تحول عند النوم وتفارق البدن بغير مباينة منه !⁋	55.0102
531.01		وكذلك فعلها أيضا في اليقظة إذا رجعت إلى ذاتها؛ فإنّه ربما رجعت إلى ذاتها⁋ورفضت الأمور الجسمانية، غير أن ذلك إنّما يبين من فعلها لئلا من أجل سكون الحواس⁋وبطلان أفاعيلها.	55.0305
532.0	καὶ μὴν¶ ἐντελεχείας οὔσης οὐδὲ ἐναντίωσιν λόγου πρὸς ἐπιθυμίας,	ونقول إنّه لو كانت النفس صورة تمامية طبيعية، لما خالفت البدن في شهواته وكثير⁋من أفاعيله،	55.1011
533.0	ἓν δὲ καὶ ταὐτὸν δι’ ὅλου πεπονθέναι τὸ πᾶν οὐ¶ διαφωνοῦν ἑαυτῷ.	بل كانت غير مخالفة له في شيء من الأشياء، وكان البدن إذا أثر فيه أثر ما⁋كان ذلك الأثر في النفس أيضا،	55.1112
534.0	Αἰσθήσεις δὲ μόνον δυνατὸν ἴσως¶ γίνεσθαι, τὰς δὲ νοήσεις ἀδύνατον.	ولكان الإنسان ذا حسائس فقط لأنّ من شأن البدن⁋الحس، وليس من شأنه الفكر والعلم والروية.	55.1213
535.0	Διὸ καὶ αὐτοὶ ἄλλην¶ ψυχὴν ἢ νοῦν εἰσάγουσιν, ὃν ἀθάνατον τίθενται.	وقد عرف ذلك الجرميون، فمن أجل ذلك⁋اضطروا إلى الإقرار بنفس أخرى وعقل آخر لا يموت.	55.1314
536.0	Τὴν¶ οὖν λογιζομένην ψυχὴν ἄλλως ἐντελέχειαν ἢ τοῦτον τὸν¶ τρόπον ἀνάγκη εἶναι, εἰ δεῖ τῷ ὀνόματι τούτῳ χρῆσθαι.	فأمّا نحن فقائلون إنّه ليست⁋نفس أخرى غير هذه النفس الناطقة التي في البدن الآن، وهي التي قالت الفلاسفة إنّها⁋انطلاشيا البدن، غير أنّهم إنّما ذكروا أنّها انطلاشيا وصورة تمامية بنوع آخر غير⁋النوع الذي ذكره الجرميون،	55.1417
536.01		أعني أنّها ليست تماما كالتمام الطبيعي المفعول، بل إنّما⁋هي تمام فاعل أي يفعل التمام. فبهذا المعنى قالوا إنّها تمام البدن الطبيعي الآلي⁋ذي النفس والقوة.⁋	55.1719
536.02		تم الميمر الثالث بحمد الله وحسن توفيقه⁋	55.2020
536.03		ولو كانت النفس تماما للبدن بأنّه بدن لما فارقته، ولما علمت الشيء البعيد،⁋ولكانت إنّما تعلم الأشياء الحاضرة كمعرفة الحواس،	55.0506
536.04		ومن شأن الحسائس أن تقبل آثار الأشياء فقط⁋فأمّا المعرفة والتمييز فللنفس.⁋	55.0809
537.0	Οὐδ’ ἡ αἰσθητική, εἴπερ καὶ αὕτη τῶν αἰσθητῶν ἀπόντων¶ τοὺς τύπους ἔχει, αὐτοὺς οὐ μετὰ τοῦ σώματος ἄρα ἕξει·	فتكون هي والحسائس شيئا⁋واحدا؛ وليس ذلك كذلك لأن النفس تعرف الشيء وإن بعد عنها وتعرف الآثار التى⁋تقبل الحسائس وتميزها كما قلنا مرارا.	55.0608

Figure 4.12: Greek against Arabic alignment with transpositions

cessing, such as pattern-matching by regular expressions and Natural Language Processing methods such as information extraction like proper names recognition. The process tries to discover hapax words or terms with a low-frequency in order to use them as inter-textual anchors for parallelization purposes. Thereafter, aligned textual chunks can be linked and manually reviewed. Furthermore, fine-grain comparisons can be performed by global alignment algorithms. Indeed, the aligner module implements the Needleman-Wunsch algorithm [81]. This is a dynamic programming solution which aligns every character in the compared sequences. In other words, the algorithm is a dynamic programming process which uses two 2-d matrix. One matrix to perform the alignment and the other for similarity evaluation.

Due to its wide applicability, the design of a software component dealing with the process of alignment emphasizes the general principles underpinning the development of a general library of components, for the textual scholarship. Two aspects characterize the process of such a design: 1) considering and putting into effect general and proven solutions to recurring design problems in a specific application context [212, 206], and 2) taking into account Application Programming Interfaces as the mean to separate services from the implementation details. In this way the aligner component, firstly developed as a specific module forOCR alignment and further generalized [252], strives to be flexible and reusable and fosters its internal structures to be as loosely coupled as possible. Consequently, the design of the aligner component exploits not only basic object-oriented mechanisms as abstraction or polymorphism, but also advanced principles and techniques as object cooperation, interface programming, separation of responsibilities and resilience to changes.

Fig. 4.13 shows the Unified Modeling Language (UML) class diagram of the aligner component.

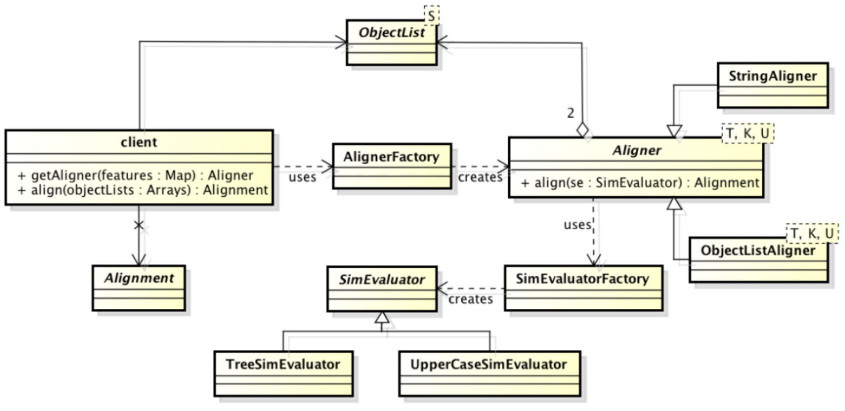


Figure 4.13: The alignment component. UML class diagram

The component design attempts to achieve a high degree of extendibility with a low degree of modification (Open-Close principle [208]). In particular, the aligner adopts two kind of design patterns: a) the Factory pattern and b) the Strategy pattern. The Factory pattern is implemented by the *AlignerFacory* class and by the *SimEvaluatorFacory* class, provides components to separate behavior from concrete implementations. Indeed, clients of the library have to work only with the

Abstract Data Types which fall outside the implementation mechanisms. Therefore, clients rely only on well-defined API which are provided and exported by the component services. In this circumstances the user of the aligner only knows the interfaces of the three external classes, namely the **Aligner**, the input data type, and the output data type (**ObjectList** and *Alignment* class, respectively). The second design pattern adopted for designing the aligner is the *Strategy* pattern. It allows to define a family of algorithms free to vary according to customizable policies. Actually, the alignment process (**ObjectListAligner**) can use a number of algorithms beyond the *Needleman-Wunsh* and a number of mechanisms to evaluate the similarity between the sequences of general entities (**TreeSimEvaluator** or **UpperCaseSimEvaluator**). For this reason, the Strategy pattern represents an important aspect in the design of the aligner APIs.

4.4 Variant reading and multi-level analysis

Once data have been coded, proofread and indexed, content enrichment is a further task. As mentioned in chapter 3 TSLib, data model handles content enrichment of the textual phenomena by annotations. This technique separates structure of the text from its content and analyses, based on a document layered view (philological, linguistic, metric, stylistic, etc). The Stand-off markup approach and Canonical Text Services (CTS) notation have been used to manage the data yielded from the document analysis (see chapter 2 section 2.4.2).

The annotation component is useful to perform collaborative annotations and arrange canonical linkage among selected text, such as named entity and authoritative resources on the web infrastructure. Annotations can be associated with chunks of text (e.g. a sentence and its translation, Fig. 3.12 in chapter 3) or with single, independent chunks of text (e.g. a single word of the original text). The Annotation framework has also been adapted and extended for specific scientific purposes [44] as illustrated in the section 4.4.1.

Finally, GUIs, Infographics concepts, and Multi-dimensional scaling techniques allow scholars to compare, to understand and to choose different variant readings (*source versions*) of a text through of computational methods [253, 254]. The multi-level annotation component has been developed within different research projects hosted at Institute for Computational Linguistics (ILC)-Consiglio Nazionale delle Ricerche (CNR).

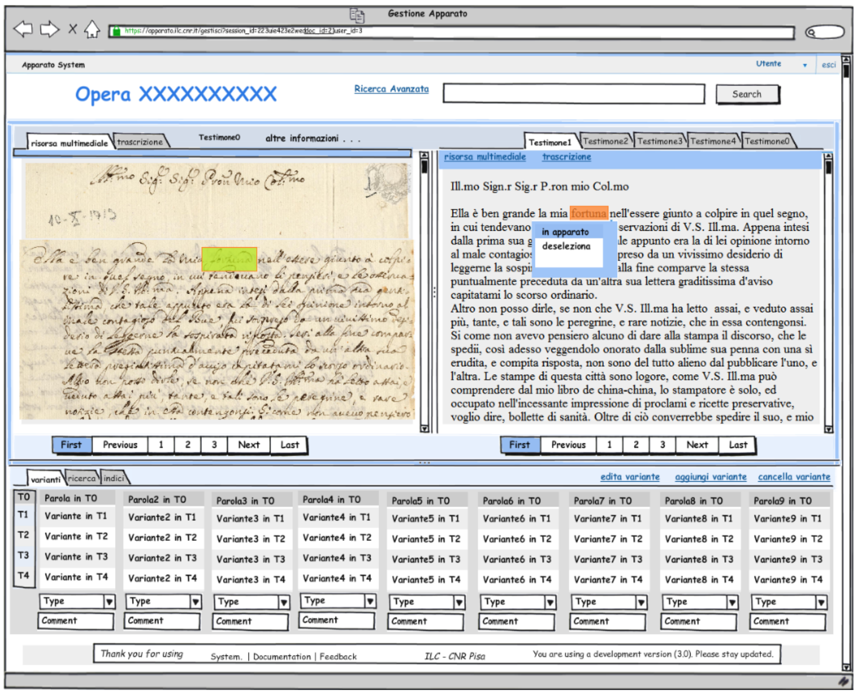
4.4.1 Variant reading annotations

Chapter 3 section 3.4 highlights the core entities of the library. In particular, the object model takes into account the collection of the manuscripts (*witnesses / tradition*) in order to reconstruct the text (*critical edition*). However, the present case study, dealing with variant readings annotations, does not solely imply the use of some kind of diff-like algorithms across the source transcriptions [87]. On the contrary, the module designed is an interactive, computer-assisted system developed in order to address scholars' needs within the editorial activity. This means that scholars have to study every word, one by one, and they have mechanisms to annotate the differences found. Fig. 4.14 shows the variant readings annotation GUI dealing with a) gathering images of each page b) transcribing the best acknowledged manuscript, c) comparing all the different versions word-by-word, to create a complete and accurate record of the differences among the versions. The annotated variant readings can be consumed by processing systems for further analysis. The computational tool can significantly facilitate the management, usability, production, and research within textual scholarship. Indeed, the method described in this work stems from a work carried out by the philological workstation project placed at ILC [35].

The user-scenario requires that scholars select what they consider to be the best witness. Afterwards, the library handles links between each word of the image and the related transcription word. These connections provide great benefit for checking the correct transcription of words which are difficult to read [25].

This approach supports scholars in recording textual-phenomena in order to compare the text conveyed by the different witnesses of the tradition. Therefore, the component can implement any algorithms (see chapter 2 section 2.3.4) to formulate outcomes according to the types established by the scholars.

The context introduced above prefers machine versus scholar interaction to artificial textual processing. On their part, the target community believes that only scholars are familiar with manuscripts and therefore that they are the sole to be entitled to identify features of different readings on the basis of which it is possible to classify variants. Later on, the variant reading module of the library leverages the features that scholars have recorded to compute similarity indexing among all the variant readings (see chapter 2 for similarity measures). As consequences, classes of variants, describing the textual tradition, provide relations of similarity weights among documents by processing the annotated text frag-



Ill.mo Sign.r Sig.r P.ron mio Col.mo

Ella è ben grande la mia fortuna nell'essere giunto a colpire in quel segno,
in cui tendevano servazioni di V.S. Ill.ma. Appena intesi
dalla prima sua g le appunto era la di lei opinione intorno
al male contagios preso da un vivissimo desiderio di
leggerne la sospi alla fine comparve la stessa
puntualmente preceduta da un'altra sua lettera graditissima d'aviso
capitatami lo scorso ordinario.
Altro non posso dirle, se non che V.S. Ill.ma ha letto assai, e veduto assai
più, tante, e tali sono le peregrine, e rare notizie, che in essa contengonsi.
Si come non avevo pensiero alcuno di dare alla stampa il discorso, che le
spedii, così adesso veggendolo onorato dalla sublime sua penna con una sì
erudita, e compita risposta, non sono del tutto alieno dal pubblicare l'uno, e
l'altra. Le stampe di questa città sono logore, come V.S. Ill.ma può
comprendere dal mio libro de china-china, lo stampatore è solo, ed
occupato nell'incessante impressione di proclami e ricette preservative,
voglio dire, bollette di sanità. Oltre di ciò converrebbe spedire il suo, e mio

T0	Parola in T0	Parola2 in T0	Parola3 in T0	Parola4 in T0	Parola5 in T0	Parola6 in T0	Parola7 in T0	Parola8 in T0	Parola9 in T0
T1	Variante in T1	Variante2 in T1	Variante3 in T1	Variante4 in T1	Variante5 in T1	Variante6 in T1	Variante7 in T1	Variante8 in T1	Variante9 in T1
T2	Variante in T2	Variante2 in T2	Variante3 in T2	Variante4 in T2	Variante5 in T2	Variante6 in T2	Variante7 in T2	Variante8 in T2	Variante9 in T2
T3	Variante in T3	Variante2 in T3	Variante3 in T3	Variante4 in T3	Variante5 in T3	Variante6 in T3	Variante7 in T3	Variante8 in T3	Variante9 in T3
T4	Variante in T4	Variante2 in T4	Variante3 in T4	Variante4 in T4	Variante5 in T4	Variante6 in T4	Variante7 in T4	Variante8 in T4	Variante9 in T4
	Type	Type	Type	Type	Type	Type	Type	Type	Type
	Comment	Comment	Comment	Comment	Comment	Comment	Comment	Comment	Comment

Figure 4.14: The variant readings annotation example

ments [255]. Moreover, the supplied data to the system can be processed and represented in some graphical form [125, 254]. For instance, this case study has investigated multi-dimensional scaling techniques for performing data processing as regards graphical visualization. This method allows text reconstruction of each manuscript, based on the recorded variant readings. In addition, all other sources in the sources collection can be automatically and dynamically obtained thanks to the apparatus (positive) which store all possible text variation.

The aforementioned process can also be used in the case of a unique document as shown in Fig. 4.9. Textual variation concerns the changes that the author of the handwritten made. For this reason both the software component and scholars need to consider these changes as variant readings [256]. This means that the changes can be managed as readings referred by different sources (see chapter 3, in particular section 3.2).

Thanks to the facsimile, it is sufficient to perform basic image processing, like zooming, to obtain high quality readings. Furthermore, adding functionality for optical filters sensitive to infrared and ultraviolet bands makes it possible to achieve better results [151, 147, 152].

4.4.2 Multi-level analysis

Digital textual scholarship starts by processing machine-actionable raw text or image of primary sources. The annotation phase, then, can begin in order to structure the logical blocks of the document as chapters, paragraphs and so on. Another important tool useful to scholars' needs is linguistic processing [257]. Computational fields like Natural Language Processing (NLP) are the best means to perform recognition, extraction and formal annotations of these kinds of information [258]. Accordingly texts have distinct levels of analysis at different units of granularity.

For instance, documents involve orthographic level (e.g. character encodings, tokenization, sentence detection), a phonological level such as sound and metrics, morphology level (word inflection and Part of Speech (POS) tagging); the words ordering encompasses syntactical levels (parsers), whereas the meaning of individual words includes the semantic level (Named Entity Recognition (NER), word meaning disambiguation). Finally, the use of words in a particular context involves the pragmatic level, while dialogues among people refer to the discourse level (co-reference resolution).

Against this background, as illustrated in the earlier chapters, the different levels of analysis can be seen as distinct layers [259, 204], ranging from orthography - taking into account the smallest units - to pragmatic and discourse, on the highest layer. It derives that linguistic information added to the content of a document simplifies its understanding and, consequently, gives the possibility for a better text reading.

In addition, document analysis allows, not only a human-agent, but also machines to be capable of processing textual content on different informative levels and at various granularity in order to manage the knowledge contained in documents. Common document analyses involve automatic linguistic metadata labeling, for instance, enriching transcriptions with part of speech or morphological information. NLP technologies include parsers and annotated corpora, automatic term extraction, information retrieval tools, and methods for automatically generating relationships to related entities (Linked Open Data (LOD)).

For example, a lemmatized and morphologically annotated document points out the lemma and the morphological features of all its words; this allows scholars to search not only the single forms in the text, but also all occurrences of a given lemma.

The analysis process has two main objectives aiming to increase the digital significance of the sources in order to: (1) build a systematic index enabling to quickly perform search operations (see section 4.2); (2) associate to each linguistic unit a morpho-syntactic and semantic information and other remarks. We can send specific requests to the system (query) by improving the information related to the text, and get the relevant results in a reasonable time. Below an example of advanced query that users might submit:

```
term_Aj NEAR_3 lem_Ak AND term_Bj NEAR_1 verb_Bk
```

This procedure means that users can search for all pairs of inter-textual excerpt containing a given term attested in a *text A*, next to a given lemma distant three words (NEAR_3) from the first term (the textual chunks which scholars are looking for), a precise term in *text B* (term_Bj) followed by a specific verb (verb_Bk).

This case study requires the definition of a workflow which includes two preliminary steps: transcribing the documents written in historical languages and translating them into Italian and English. Transcriptions and translations are encoded by using consistent markup TEI-XML. Hence, as Fig. 4.15 illustrates the Multi-level analysis process, encompasses several phases: 1) sentence detection 2) tokenization 3) linguistic annotation, i.e. morphological analysis as well lemmatization 4) lexical annotation 5) semantic annotation. Starting from the assumption that a text has to be linguistically enriched, the TSLib module starts by dividing it into sentences. The point where the sentence starts and where it ends can be previously marked-up in the structured document or can be automatically detected by means of suitable automatic techniques such as Machine Learning or regular pattern recognition. After that, each sentence is divided into words (tokens). The task of identifying the tokens is called “tokenization”. Sentences and tokens are the input units for further analysis, like part-of-speech tagging.

Sentence detection, tokenization, linguistic annotation and named entity recognition are all well-known NLP tasks. The lexical and semantic annotation exploit ontology schemes inspired by already existing ontologies or conceptualization schemes as explained in chapter 2. It ensues that such tools need to support both document markup and information extraction.

The textual scholarship library attempts to meet the aforementioned require-

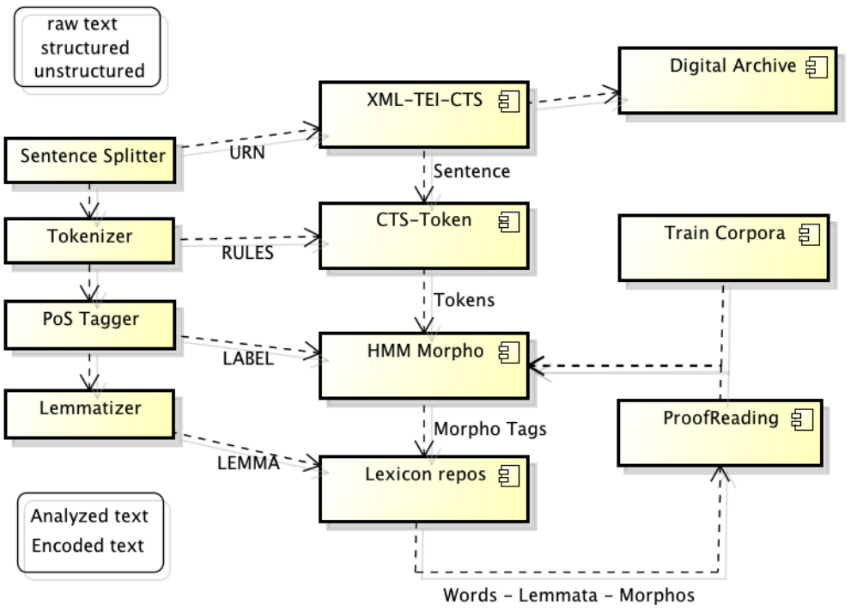
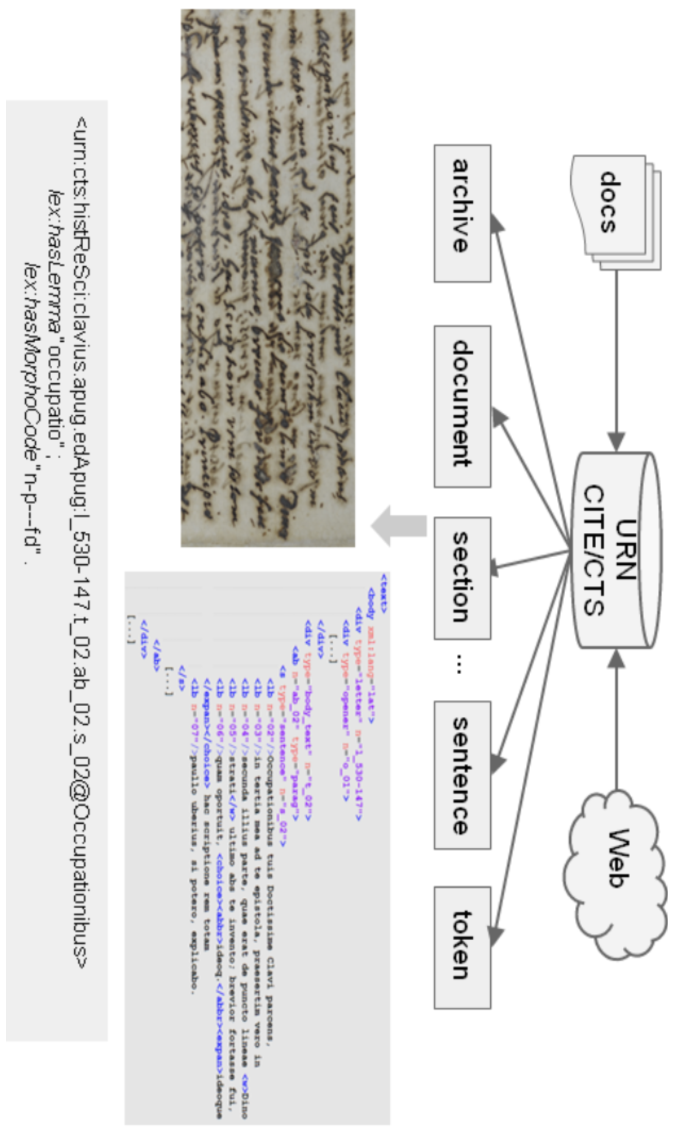


Figure 4.15: TSLib module for text analysis and segmentation

ments. Indeed, the library encompasses a sentence splitter and a tokenizer both assigning an identifier to each source element (see chapter 3, specifically section 3.4) in the digitized texts. In particular CTS compliant structure (see chapter 2) guarantees a global protocol and a citational schema by adopting unique Uniform Resource Name (URN) identifiers. It is worth noting that CTS specifies a protocol for citation purposes, consequently the policies for data persistence are decoupled from the serialization schema adopted. This allows the module to build unique identifiers for each word in the documents, together with the citation scheme. Fig. 4.16 shows the multi-layer annotation component with CTS-URN notation. The TEI-XML encoded texts are broken into structural sections with divisions into blocks, paragraphs, sentences, lines in order to construct the levels of the URN.

The module implements the canonical citation scheme by using the hierarchical structure of the document. Indeed, the structure of the digital sources exploits the TEI XML basic elements to provide CTS compliant notation (see chapter 2). This means that the markup encodes chapters, sections, sentences and abbreviations in well-defined elements in order to construct the URN identifier. Two well-known tools for producing a lemmatization and morphological analysis of ancient texts



```
<text>
  <body xml:lang="lat">
    <div type="letter" n="l_530-147">
      <div type="opener" n="o_01">
        [...]
      </div>
      <div type="body_text" n="t_02">
          <ab n="ab_02" type="parag">
            <s type="sentence" n="s_02">
              <lb n="02"/>Occupationibus tuis Doctissime Clavi parcens,
              <lb n="03"/>in tertia mea ad te epistola, praesertim vero in
              <lb n="04"/>secunda illius parte, quae erat de puncto lineae <w>Dino
              <lb n="05"/>strati</w> ultimo abs te invento; brevior fortasse fui,
              <lb n="06"/>quam oportuit, <choice><abbr>ideoq.</abbr><expan>ideoque
              </expan></choice> hac scriptione rem totam
              <lb n="07"/>paullo uberius, si potero, explicabo.
            </s>
            [...]
          </ab>
      </div>
    [...]
```

```
<urn:cts:histReSci:clavius.apug.edApug:l_530-147.t_02.ab_02.s_02@Occupationibus>
        lex:hasLemma"occupatio" ;
                lex:hasMorphoCode"n-p---fd" .
```

Figure 4.16: The multi-layer annotation component

are: *Morpheus* [260] and *CHLT-LEMLAT* [158] software systems.

On the one hand, the Morpheus analyzer has been implemented within the Perseus project and it supports Greek and Latin. On the other hand, CHLT-LEMLAT has been developed at the ILC-CNR and it supports Latin texts. The ongoing multi-level analysis component applies machine learning to estimate the correct morphological analyses for a given word in a given sentence. In case scholars disagree with the machine outcomes, they can review the analysis for proofreading. This software provides both the most probable (following the three-gram Hidden Markov Model (HMM) implemented in the *HUNPOS* tool) and all the possible morphological analyses of an input word. The main advantage of automatic annotation is its time-saving feature and scalability. Indeed, automatic methods produce systematic annotations on a statistical bases, which are, moreover, easier to detect and correct than manual mistakes.

The multi-level analysis case study aims at developing a component for the TSLib, according to the data model illustrated in chapter 3. It deals with information extraction and textual data organization, which is implicitly present in the text. In particular, the component allows both scholars and machines to process the content of primary sources thoroughly, in order to better understand their textual data. This means that the component is able to systematically manage textual entities and to establish relationships among them. In addition, the component exposes textual concepts by consulting domain ontologies and semantically structured lexicons. Finally significant entities are linked to a Named Entity Repository and exported to the cloud following the LOD principles [164].

As Fig. 4.17 shows, the components of the linguistic module provide functionality to automatically divide the content of the document in sentences and tokens, both of which have their own CTS unique identifier. Especially, The TSLib lemmatization module involves several components: the source parser of electronic texts adapting Tika API; the *sentence splitter* through the XML processing and CTS notation; the *word tokenization* by means of rule-based process with CTS Uniform Resource Identifier (URI); the *HMM POS tagger* by exploiting *HunPos* tool; the *Lemmatizer* by accessing large lexicons of word forms; the *TreeBanks* handler to learn the statistical classifiers; and the *Proofreader* component allowing scholars to correct the erroneous outcomes.

The tokenization process and its outcomes follow the same rules introduced in the indexing section (see section 4.2). Thereafter, the actual component in TSLib for linguistic analysis includes an HMM POS tagging and a look-up process for lemmatization. This means that TSLib is able to process content and annotate each token based on the abstract data model illustrated in chapter 3. It derives that the document includes: a) CTS URN (*version*), b) token detection

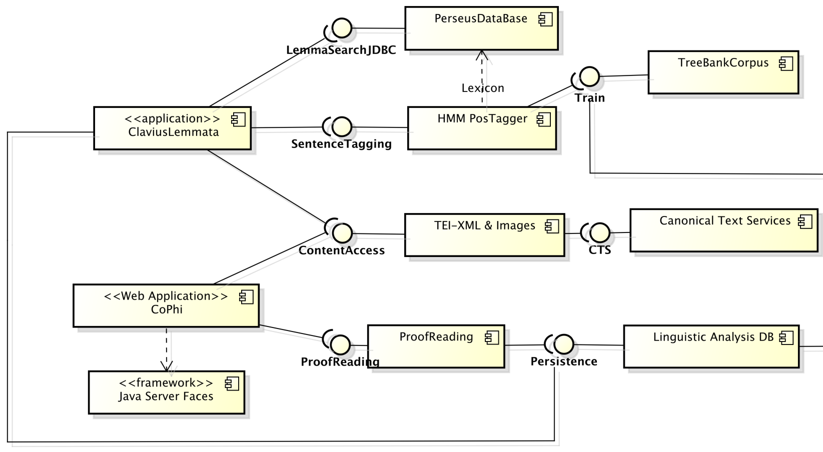


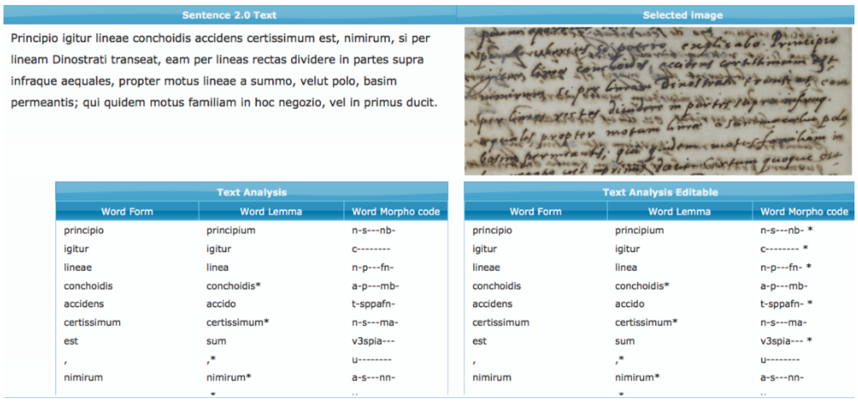
Figure 4.17: The TSLIB lemmatization module

(*granularity*) c) character sequence start and end (*position*), d) Grammatical Category (*morphological layer*), e) morphological features (*morpho-syntactic layer*), d) lemma (*lemmatization layer*).

The lemmatization component of the TSLib has been implemented as a supervised statistical classification problem with a lexicon for monitoring the word-form and the annotated lemma. For this reason, the training set adopted is a human annotation resource. Indeed, the basic idea behind automatic classification is that it can produce annotations similar to human annotators if the text to be annotated is sufficiently similar to the texts used for training the component (see chapter 2). The nature of the language is ambiguous, therefore a token which is processed without contextual information has different possible interpretations. Thus, Machine Learning techniques for NLP always provide contextual information, in the case of the prototype here exposed, the HMM came with n-grams of context to better tune the probability of the tag sequence related to the supplied tokenized-sentence.

$$\arg\max_{t_1 t_T} P(t_{T+1}|t_T) \prod_{i}^{T} P(t_i|t_{i-1}, t_{i-2}) P(w_i|t_{i-1}, t_i)$$

Hence, the morphological step aims at tagging the part-of-speech for each token through a statistical algorithm using HMM Machine Learning technique. At the end of this phase, each token has a morphological label which determines its most likely morphological interpretation. Fig. 4.18 shows the format of the morphological code. It is derived from the morphological label used in the TreeBank of the Perseus project.



Principio igitur lineae conchoidis accidens certissimum est, nimirum, si per lineam Dinostrati transeat, eam per lineas rectas dividere in partes supra infraque aequales, propter motus lineae a summo, velut polo, basim permeantis; qui quidem motus familiam in hoc negozio, vel in primus ducit.

Text Analysis

Word Form	Word Lemma	Word Morpho code
principio	principium	n-s---nb-
igitur	igitur	c--------
lineae	linea	n-p---fn-
conchoidis	conchoidis*	a-p---mb-
accidens	accido	t-sppafn-
certissimum	certissimum*	n-s---ma-
est	sum	v3spia---
,	,*	u--------
nimirum	nimirum*	a-s---nn-

Text Analysis Editable

Word Form	Word Lemma	Word Morpho code
principio	principium	n-s---nb- *
igitur	igitur	c-------- *
lineae	linea	n-p---fn- *
conchoidis	conchoidis*	a-p---mb-
accidens	accido	t-sppafn- *
certissimum	certissimum*	n-s---ma-
est	sum	v3spia--- *
,	,*	u--------
nimirum	nimirum*	a-s---nn-

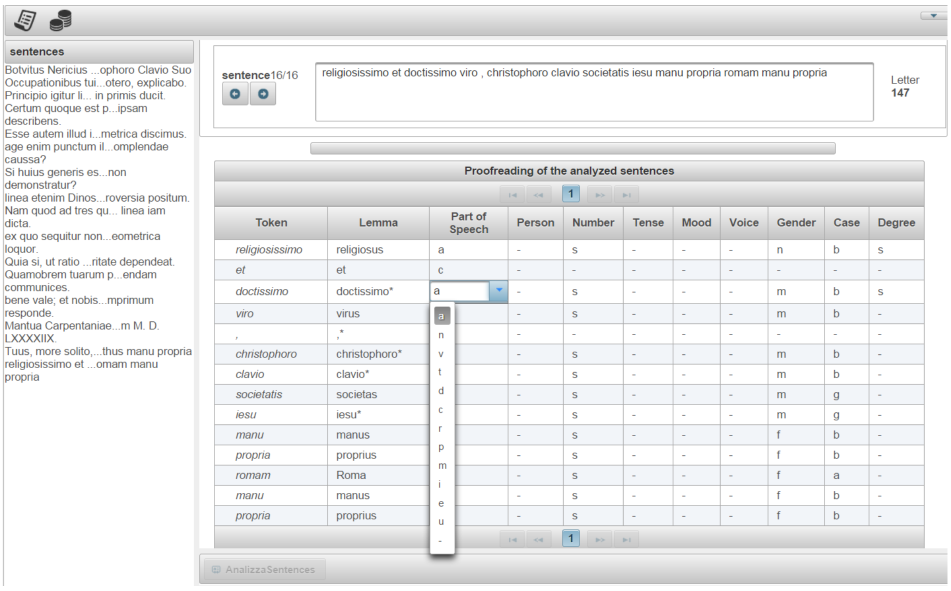
Figure 4.18: Example of linguistic annotation to a sentence

Additional elements of linguistic nature are combined to each processing unit (token) and made available for scholars to use this new information. The data enrichment process produces a structured information stored in a look-up table where each token has a list of associated values. The system also manages the links to the document in the collection. This linguistic annotation module exploits the HunPos tool. It is one of the most efficient and used POS tagger for language with complex morphology [165, 261].

The training set of TSLib for this case study uses the Latin TreeBank, which has been manually annotated from the Perseus Project [262, 166]. The Treebank, which is a collection of sentences with morphological and syntactic level annotation, consists of 53.143 tokens and represents the syntactic trees of the sentences therein. Moreover, The training phase involves 3.474 sentences and 438 different tags. Consequently, Treebanks are worthy resources adopted by computational tasks as a basis for developing automatic methods for annotations.

Thereafter, lemmatization can benefit from POS tagging. In fact, the grammatical category of speech, featuring nouns, verbs or adjectives, is used to disambiguate the annotation.

Token streams do not always have a single correct analysis, but different labels can be possible. Indeed, the system allows scholars to provide feedback on tagged and lemmatized words. For example, scholars or students can annotate whether a grammatical category in a selected context is correct, whether a token is a correct word with the correct lemma and with the right features label; whether a certain lemma is a lexical definition in the domain ontology, rather than a false positive.



Botvitus Nericius ...ophoro Clavio Suo
Occupationibus tui...otero, explicabo.
Principio igitur li... in primis ducit.
Certum quoque est p...ipsam describens.
Esse autem illud i...metrica discimus.
age enim punctum il...omplendae causa?
Si huius generis es...non demonstratur?
linea etenim Dinos...roversia positum.
Nam quod ad tres qu... linea iam dicta.
ex quo sequitur non...eometrica loquor.
Quia si, ut ratio ...ritate dependeat.
Quamobrem tuarum p...endam communices.
bene vale; et nobis...mprimum responde.
Mantua Carpentaniae...m M. D. LXXXXIIX.
Tuus, more solito,...thus manu propria
religiosissimo et ...omam manu propria

religiosissimo et doctissimo viro , christophoro clavio societatis iesu manu propria romam manu propria

Proofreading of the analyzed sentences

Token	Lemma	Part of Speech	Person	Number	Tense	Mood	Voice	Gender	Case	Degree
religiosissimo	religiosus	a	-	s	-	-	-	n	b	s
et	et	c	-	-	-	-	-	-	-	-
doctissimo	doctissimo*	a	-	s	-	-	-	m	b	s
viro	virus		-	s	-	-	-	m	b	-
,	,*		-	-	-	-	-	-	-	-
christophoro	christophoro*		-	s	-	-	-	m	b	-
clavio	clavio*		-	s	-	-	-	m	b	-
societatis	societas		-	s	-	-	-	m	g	-
iesu	iesu*		-	s	-	-	-	m	g	-
manu	manus		-	s	-	-	-	f	b	-
propria	proprius		-	s	-	-	-	f	b	-
romam	Roma		-	s	-	-	-	f	a	-
manu	manus		-	s	-	-	-	f	b	-
propria	proprius		-	s	-	-	-	f	b	-

Figure 4.19: The TSLib Human correction module

Fig. 4.19 shows the GUI for the review process. TSLib performs statistical annotation. Consequently, mistakes can occur and they have to be corrected by hand (using a suitable proofreading GUI). The combination of high-speed automatic annotation and high-quality human corrections is the current solution for textual scholarship multi-layer annotations.

The process is divided in four steps. 1) The module detects the sentence from the object representation of the document which is the context for the HMM morphological analyzer module. 2) the analyzer component generates its analysis for each word-form of the sentence. (Accuracy of the automated disambiguation stands at 60% at the time of writing). 3) the lemmatizer evaluates the dictionary entries (and the most frequent in the collection is kept if the word is lexically ambiguous), 4) Scholars or students evaluate the lemma matched to each word-form and the morphological analysis. Indeed, each word sense is appropriate to a given word in a given sentence (context).

Chapter 5

Conclusion and Perspectives

The research here offered has outlined a number of design aspects and concrete case studies concerning the Digital Humanities field. This thesis has mainly highlighted the computational issues and methodological principles involved in the design of a modular library for digital textual scholarship (called throughout TSLib). Such a work embraces different primary issues, including source acquisition, text encoding, multi-level text analysis, annotations, collaborative commenting, and the production of critical editions. Therefore, the aim of TSLib is to support scholars by handling primary and secondary sources as well as guaranteeing its long-term maintenance.

As illustrated in the previous chapters, the attention on the fast-moving Digital Humanities field has been increasing from different research fields. In particular: a) computer scientists focus on developing or improving algorithms for document and text processing such as phylogenetic problems, multi-version issues, OCR accuracy, indexing, etc.; b) computer engineers work on designing and implementing effective and reusable tools and virtual environments dealing with scholar requirements such as APIs design, standardization of communication protocols, integration mechanisms; and finally c) traditional and digital scholars leverage computation outcomes and electronic documents in order to improve their understanding of the work under investigation.

The research has shown that, at the time of writing, shared methodological frameworks and flexible tools are still an open issue for this particular field. Actually, this thesis has highlighted that the current textual scholarship applications are too often project-oriented, therefore they lack in generalization and abstraction. This means that many initiatives do not adequately consider reusability and evolutionary perspectives both in terms of software artifacts and data sets. Consequently, actual textual scholarship outcomes mainly solve limited problems or meet restricted domain requirements such as markup practices, publishing processes, or collation issues. In addition, computational methods and tools exploited in literary computing generally follow other disciplines such as computational linguistics or computational biology. Indeed, in those scientific areas, theoretic models, computational methods, and community best practices are now consolidated and well-established.

This work has also argued that, contrary to usual approaches, the textual scholarship community has to consider well-known engineering strategies in its computational researches for effectively enhancing the area of literary studies in the digital age. As a matter of fact, new Digital Humanities initiatives have begun to discuss on how to set up effective processes for the suitable design and development tools for literary studies.

In other words, one of the aims of this thesis has been to encourage a community-driven research activity for the specification and the implementation of reusable textual scholarship components.

The design of such an artifact has to encompass several points including a community oriented perspective, a use-case process, UML modeling, and API standardization. Furthermore, the work has discussed basic requirements of the target domain which the TSLib deals with. These requirements describe texts that can be written on multiple supports, conveying multiple versions of the same work which can have multiple hierarchies and interpretations at multiple levels of granularity. The results of this analysis have led to the representation of textual units as composite entities defined by an array of four properties: <version, granularity, position, layer>.

Since application programming interfaces handle data and behavior of the digital representation of the sources, software artifacts in literary computing have to be designed in terms of components. In this way, each component provides specialized services by means of standard and common interfaces (API). In turn, inter-components communication has to be designed on community-based APIs, so that they can totally decouple the interface from internal details (information hiding). Doing so, actual implementation of the components is totally managed by the service providers. Afterwards, only the API specification has to remain open and community driven.

Practical case studies, derived from the work conducted in the context of funded national and international projects (mainly ERC and PRIN projects), have pointed out that the combination of a bottom-up (starting from specific needs) and a top-down approach (starting from abstract models) promotes the enhancement of the design solutions. Moreover, refactory strategies, as defined in the Agile processes, help in obtaining improvements and in meeting new requirements. Thus, this research is an attempt towards a methodological approach for designing and developing literary tools led by an active and dynamic international community. This can be done by adhering to and improving the framework described throughout the work.

The component devoted to manage the services of the library allows providers and clients to plug-in new modules, keeping the TSLib open for extension but closed for modification. Hence, the fundamental artifacts, which are conceived as components, have been organized (at present) in Java packages mapping the core modules. Up to now, seven components constitute the core of the library. These encompass the requirements gathered from the textual scholarship community and involve the management of 1) the textual sources, 2) the facsimile representation, 3) the editing phase, 4) the analysis at different levels of processing, 5) the relationship and linkage mechanisms, 6) the indexing features, and finally 7) the visualization and rendering of the data.

Therefore, the goal of this research is to implement all these components by means of coherent APIs which will be released in the form of a software library. In fact, the effort is devoted to achieve an evolvable specification of the application programming interfaces along with the description of a convenient data structure. The data model serves as the basis for concrete objects which instantiate abstract data types that meet the API specification. Indeed, differently from current habits in Digital Humanities, the TSLib tries to abstract its data model from file formats by designing suitable ADTs and APIs. By doing so, clients can use the TSLib and exploit all functions of the core API.

However, at the time of writing, the domain specification that concerns the digital humanities scholars is only partially accomplished. This is mainly due to the incomplete formalization of the discipline and consequently to the fragmentary gathering of its requirements.

Acknowledging the importance of the open-source policy in modern research, the ongoing work is publicly available for the community. Moreover, the TSLib promotes the publishing of the scholars works as linked open data (LOD), promoting the circulation of knowledge regarding cultural heritage documents and texts. Accordingly, the model of the data in the TSLib is LOD compliant, this means that the URIs are stable and properly dereferenceable by standard citational protocols.

Summing up, the library of components described so far attempts to fill a gap in the field of research works concerning new technologies applied to literary studies. Nowadays, although the implementation of the library is just a prototype, several projects have exploited its model. As the case studies chapter has discussed, the TSLib model already guarantees a large and simultaneous access to the information pertaining to manuscripts and other forms of textual and linguistic sources.

Nevertheless, the work is still far from being mature and completed, consequently, it has to be improved in many ways. For instance, any further work should involve the production of the necessary documentation. Indeed, the documentation provides a deeper insight into how a component may be used in typical contexts and for typical usage. In terms of future research and development, it is worth emphasizing that an editing component for scholars could be plugged into the system. This would support the creation of a different edition of the same text and improve the comprehension of its history. In addition, the TSLib fosters digital humanists to think in term of services and abstractions. Actually, two levels of accessibility have to be pursued: 1) textual scholarship tools need to be accessible for scholars, 2) the design process and the implementation methodologies of those tools need to be shared.

In conclusion, despite a number of pioneering researches, digital scholar applications are relatively undeveloped if compared to other textual software like those in natural language processing, text mining, and bioinformatics. However, digital textual scholarship is a computational field in rapid evolution. Hence, textual tools devoted to solve scholar needs (with or without technological skills) must provide ad hoc capabilities developed within a modular library for textual scholarship.

Unfortunately, time constraint has limited a deeper analysis of the target domain and the proper implementation of the design methodologies illustrated in the previous chapters. The wish is to have the opportunity to continue the work which has been presented in this thesis.

Chapter 6

Acronyms

ASCII	American Standard Code for Information Interchange	32
API	Application Programming Interface	109
ADT	Abstract Data Type	78
ANN	Artificial Neural Network	55
ALPS	Application Level Placement Scheduler	115
BMP	Basic Multilingual Plane	32
CLARIN	Common Language Resources and Technology Infrastructure	22
CTS	Canonical Text Services	133
CMDI	CLARIN Metadata Infrastructure	31

CRUD	Create, Read, Update, Delete	69
CITE	Collections, Indexes, Texts, Extensions architecture	21
CAS	Common Analysis Structure	48
CNR	Consiglio Nazionale delle Ricerche	133
CLE	Cray Linux Environment	115
CSV	Comma Separated Values	105
CCSL	CMDI Component Specification Language	40
CHI	Cultural Heritage Imaging Organization	53
DARIAH	Digital Research Infrastructure for the Arts and Humanities	22
DH	Digital Humanities	61
DTD	Document Type Definition	34
DCMI	Dublin Core Metadata Initiative	37
DC	Dublin Core	37
DAO	Data Access Object	106

DOM	Document Object Model	121
DM2E	Digitised Manuscripts to Europeana	24
DB	Data Base	30
DOC	document	105
DCIF	Data Category Interchange Format	40
EDM	Europeana Data Model	24
ESE	Europeana Semantic Elements	38
XSLT	eXtensible Stylesheet Language Transformations	121
ESTS	European Society for Textual Scholarship	19
FRBR	Functional Requirements for Bibliographic Records	83
FSF	Free Software Foundation	18
GDocs	Google Documents	18
GUI	Graphical User Interface	122
GATE	General Architecture for Text Engineering	47

HMM Hidden Markov Model 140

hOCR HTML-based Markup for OCR 116

HTML HyperText Markup Language 116

HTTP HyperText Transfer Protocol 24

HERA European Research Area 24

IEC International Electrotechnical Committee

ISO International Organization for Standardization 23

ILC Institute for Computational Linguistics 133

IFLA International Federation of Library Associations 38

IDE integrated development environment 91

JSF JavaServer Faces Technology 62

JSON JavaScript Object Notation 105

LOD Linked Open Data 136

ML Machine Learning 46

MPI	Message Passing Interface	115
MVC	Model View Controller	106
MVD	Multi-Version Document	36
NLP	Natural Language Processing	136
NER	Named Entity Recognition	136
OHCO	Ordered Hierarchy of Content Objects	83
OCR	Optical Character Recognition	112
OAI	Open Archive Initiative	40
OA	Open Annotation Data Model	85
ORE	Object Reuse and Exchange	40
OAIS	Open Archival Information System	40
OAC	Open Annotation Collaboration	23
OASIS	Organization for the Advancement of Structured Information Standards	48
OWL	Web Ontology Language	

OSI	Open Source Initiative	18
PMH	Protocol for Metadata Harvesting	40
POS	Part of Speech	136
PBS	Portable Batch System	115
PDF	Portable Document Format	105
RDF	Resource Description Framework	77
ROI	Region of Interest	41
SAWS	Sharing Ancient Wisdom	24
SPI	Service Provider Interface	95
SVM	Support Vector Machine	55
SofA	Subject of Analysis	48
SLES	SUSE Linux Enterprise Server	115
SAX	Simple API for XML	122
SSO	Single Sign On	23

SVG	Scalable Vector Graphics	29
SGML	Standard Generalized Markup Language	34
SPARQL	Simple Protocol and RDF Query Language	
SW	Semantic Web	24
TUSTEP	Tübingen System of Text Processing	27
TXSTEP	Tübingen XML-based scripting language for scholarly text data processing	27
TSLib	Textual Scholarship Library	109
TEI	Text Encoding Initiative	119
TE	Term Extraction	53
UCS	Universal Character Set	32
UTF	Unicode Transformation Format	32
UNICODE	Unique, Universal, and Uniform character enCoding	31
UML	Unified Modeling Language	132
URN	Uniform Resource Name	138

URI Uniform Resource Identifier 140

UIMA Unstructured Information Management 48

VSM Vector Space Model 51

XML The eXtensible Markup Language 119

XMI XML Metadata Interchange 48

XSL-FO Extensible Stylesheet Language Formatting Objects

XSD XML Schema Definition 34

XHTML eXtensible HyperText Markup Language 34

Chapter 7

Acknowledgements

This work is the result of a close collaboration among many people coming from different backgrounds and institutions. First of all I am thankful to the Institute for Computational Linguistics, in particular to Dr. Andrea Bozzi for his continuous encouragements and for supporting me. I am also grateful to Dr. Simonetta Montemagni, current director of the Institute, for her kindness. Moreover I express my sincere gratitude to my supervisor, Prof. Francesco Marcelloni, for the insightful comments and help.

My biggest thanks go to my supervisors, Dr. Emiliano Giovannetti and Dr. Federico Boschetti. I am touched by the way they have supported me and taken care of my work.

Furthermore, I am grateful to Mariarosaria Finelli for her extraordinary closeness, spiritual and material support. I am thankful to my whole family: parents, sister, nephew, uncles, cousins and grandparents (to those here and to who can not be here).

My work would not have been possible without the support of many, many friends and co-workers. Thanks a lot to Ouafae Nahli (second mum, she has been extraordinary), Simone Marchi (first brother), Riccardo Del Gratta (LaTeX guru), Marion Lamé (bibliographic superhero), Davide Albanesi (13 room chief), Andrea Bellandi (13 room mind), Giulia Benotto (13 room patience), Marianne Reboul (great supporter), Rosanna Martucci (English ghost writer), Genoveffa Maselli (first English reviewer), Francesca Rispoli (second English reviewer), Felice Dell'Orletta (NLP Italian boss and supreme football player), Francesca Murano

(Saussure expert and a dear friend), Achille Felicetti (expert and great guy), Luca Pesini (XML guru! and a real Indo-Europeanist), Anas Fahad Khan (shrewd logician and friend). Paolo Pegoraro, Daniela and Louay, and don Angelo Colacrai for their help and encouragement.

The results yielded in this thesis have also been possible thanks to the financial support of several European and Italian projects that have funded it. Among these: 1) The ERC project Ideas, Advanced Grant 249431 "Greek into Arabic", supported by the European Research Council and led by Cristina D'Ancona. 2) The PRIN2008 project "Per un'edizione digitale dei manoscritti di Ferdinand de Saussure" led by Daniele Gambarara and Maria Pia Marchese. 3) The PRIN2010/11 "Memoria poetica e poesia della memoria" led by Paolo Mastandrea. 4) the PRIN "Sistema di Filologia Computazionale per la gestione di immagini e testi digitali, l'indicizzazione e la produzione di apparati critici" led by Andrea Bozzi.

Last but not the least, a great thought to Maria, mother of God and our mother. She always accompanies me!

Finally, I would like to thank the people who have been close to me during this research, and whom I forgot to mention. My apologies for that, but my head has not been working well, lately!!

Afterword

The following text written in Italian comes directly from the pen of Father Busa. One of my best friends inadvertently sent me this valuable piece of evidence. It seems that Computer engineers and philologists have a long tradition in common..

Dall'Introduzione a Fondamenti di informatica linguistica - Art. 2: caratteristiche del corso

0006. A proposito del dialogo tra filologi e informatici, ecco alcune osservazioni dettate dall'esperienza. Già tra linguisti e filologi il discorrere è reso talora difficoltoso dalle stesse terminologie linguistiche, troppe volte diverse e fluide, ma il dialogo tra filologi e informatici risulta ancor più problematico. La formazione mentale del matematico, dell'ingegnere, dell'economista è radicalmente diversa da quella del filologo, e di conseguenza altrettanto diverse sono le terminologie e le concentrazioni di interessi mentali. Nei cultori delle scienze esatte il metodo è prevalentemente deduttivo, la logica è cartesiana e geometrica: “*tanto mi dà tanto*”. Un informatico ingegnere inclinerà a estendere alle parole la univoca omogeneità dei numeri: in un discorso, per esempio, misurerà, tutte assieme le frequenze di parole come se queste parole avessero tutte lo stesso peso, quasi un sacchetto di fagioli: metafora cui mi porta la loro distinzione in due cotiledoni, significante e significato... Non gli passa per la mente che i tipi di semanticità, quelli cioè che intaccano proprio questo rapporto tra segno e concetto, sono tanti e talmente diversi da far riscontro alle diversità esistenti nella scala periodica degli elementi di Mendelejeff... per esempio preposizioni, pronomi personali o deittici, verbi e nomi comuni di oggetti sono parole più diverse tra loro di quanto non lo siano metalli e gas e terre rare... Inoltre un informatico, in altre sedi, solitamente prepara programmi dei quali molte e ripetute esecuzioni ammortizzeranno il costo della programmazione. In linea di principio, in ricerche di linguistica informatica, avviene il contrario: ogni programma viene usato una volta sola per un’operazione il cui *output* dovrà subito essere *input* di altro e diverso programma. Quando invece il filologo chiedesse di far girare lo stesso programma, spesso sarà perché vuole farvi cambiare qualche istruzione: cosa che può risultare irritante all’informatico. Uno stesso programma entro uno stesso lavoro verrà probabilmente fatto rigirare identico solo dopo avvenute doverose correzioni al suo *input.* Altri lavori, pur identici di procedura, il più delle volte richiederanno variazioni di formato dei dati di ingresso, al che recalcitreranno quei programmatori che non abbiano a mano programmi “generalizzati” o “tabelle esterne” per veloci e facili adattamenti. E ancora, molte delle nostre operazioni sono a *batch*, cioè a “infornate di pani”, tante e consecutive: decine di migliaia e centinaia di migliaia di voci da elaborare una dopo l’altra. Nei computer che devono servire interattivamente contemporaneamente a molti terminali, i nostri *batch* creano impazienze... Altrettanto quando con migliaia e migliaia di righe teniamo occupata la *high speed printer...* Nei calcoli matematici e contabili, l’*input* e *output* sono piuttosto di piccole dimensioni, mentre le elaborazioni possono essere complesse e lunghissime. In informatica linguistica è vero l’opposto: poche e semplici operazioni su enormi *input* e con enormi *output.* A Venezia nel 1975-1976 ho dovuto far sostare su 4 campi a cascata un file di 6 milioni di *records* di 350 *bytes*, uno di 2,5 milioni di *records* di 450 *bytes* e altri due rispettivamente di 600.000 e 1.200.000 *records* lunghi altrettanto... Ve lo immaginate? E la linguistica informatica si augurerebbe di ritrovarsi spesso in tali congiunture...

0007. Fa parte del “tono pratico” del mio corso l’addestramento a quella paziente e indefessa perseveranza che richiede l’impiego dei computer: è necessario che l’allievo si tempri al frequente imprevisto del computer che si inceppa, dell’operatore che resta a casa, del *bug*, cioè errore di *input* o di programma - parlo di quello raro - che non “salta fuori” se non dopo aver elaborato senza intoppi migliaia di righe innocenti. L’informatica linguistica va affrontata come una corsa a ostacoli: essa consegna il suo premio alla fine; una volta terminato tutto e bene, ci si rende conto che si è reso un grosso servizio: testo e lessici elettronici restano validi e disponibili per tutti e sempre e la fatica impiegatavi resta utile a tutti e per sempre. Il grosso del corso è dunque pratico, su una sola linea di sviluppo, che è però quella fondamentale e necessaria e iniziale, per forza di natura di cose. Di fronte ai miti giornalistici del cervello elettronico, di fronte ai luccichii psichedelici dei paroloni, di fronte al fascino esotico degli acronimi tecnici quali ad esempio RAM, ROM, LAN (*random access memory*, *read only memory*, *local area network*) da una parte, e dall’altra di fronte all’accelerato evolversi delle tecnologie, per il quale quelle di ier l’altro sono già obsolete, mentre la “migliore” e la più “attuale”, se pur si afferrano, sgusciano e scappano subito via di mano, dico: “Comincia a fare di fatto tu, personalmente, un po’ di informatica oggi; comunque comincia con poco, perché ogni vita nasce piccola. Continua con la semplicità tranquilla di chi sa che passo dopo passo si arriva. Quando avrai raggiunto come primo obiettivo la cartografia del lessicologico d’un testo, potrai parlare con cognizione di causa e decollare alzandoti verso ogni altro e nuovo spazio informatico”.

Bibliography

[1] R. Siemens, M. Timney, C. Leitch, C. Koolen, A. Garnett *et al.*, "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*, vol. 27, no. 4, pp. 445–461, 2012.

[2] J. McGann, "From text to work: Digital tools and the emergence of the social text," *Variants: The Journal of the European Society for Textual Scholarship*, vol. 4, pp. 225–240, 2005.

[3] A. Bozzi, "Edizione elettronica e filologia computazionale," in *Fondamenti di critica testuale*, ser. Manuali, A. Stussi, Ed. Bologna: Il Mulino, 2006, ch. 9, pp. 207–232.

[4] P.-E. Portier and S. Calabretto, "DINAH, a Philological Platform for the Construction of Multi-structured Documents," in *Proceedings of the 14th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Glasgow, UK.* Berlin, Heidelberg: Springer-Verlag, September 2010, pp. 364–375. [Online]. Available: http://dl.acm.org/citation.cfm?id=1887759.1887808

[5] A. Bozzi, "Computer-assisted scholarly editing of manuscript sources," in *New publication cultures in the humanities: exploring the paradigm shift*, P. Davidhazi, Ed. Amsterdam: Amsterdam University Press, 2014, pp. 99–115. [Online]. Available: http://www.oapen.org/record/515678

[6] P. Robinson, "Towards a scholarly editing system for the next decades," in *Sanskrit Computational Linguistics*, ser. Lecture Notes in Computer Science, G. Huet, A. Kulkarni, and P. Scharf, Eds. Springer

Berlin Heidelberg, 2009, vol. 5402, pp. 346–357. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00155-0_18

[7] ——, "Towards a theory of digital editions," *Variants*, no. 10, pp. 105–131, 2013.

[8] F. Tomasi and F. Vitali, "Collaborative Annotations in Shared Environments: Metadata, Vocabularies and Techniques in the Digital Humanities (DH-CASE 2013)," in *Proceedings of the 2013 ACM Symposium on Document Engineering (DocEng), Florence, Italy.* New York, NY, USA: ACM, September 2013, pp. 283–284. [Online]. Available: http://doi.acm.org/10.1145/2494266.2494323

[9] P. Schmitz, L. Pearce, and Q. Dombrowski, "DH-CASE II: Collaborative Annotations in Shared Environments: Metadata, Tools and Techniques in the Digital Humanities," in *Proceedings of the 2014 ACM Symposium on Document Engineering (DocEng), Fort Collins, Colorado, USA.* New York, NY, USA: ACM, September 2014, pp. 211–212. [Online]. Available: http://doi.acm.org/10.1145/2644866.2644898

[10] M. Terras and G. Crane, Eds., *Changing the Center of Gravity: Transforming Classical Studies through Cyberinfrastructure*, ser. Digital Technologies and the Ancient World. Piscataway: Gorgias Press, March 2010, vol. 4.

[11] W. McCarty, *Humanities Computing.* Palgrave Macmillan, 2005.

[12] I. Lancashire, "Computers in the linguistic humanities: Overview," in *Encyclopedia of Language and Linguistics*, 2nd ed., K. Brown, Ed. Oxford: Elsevier, 2006, pp. 793–809. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B0080448542009809

[13] M. Terras, J. Nyhan, and E. Vanhoutte, *Defining Digital Humanities: A Reader.* Farnham Surrey: Ashgate, 2013.

[14] M. K. Gold, Ed., *Debates in the digital humanities.* Minneapolis, MN: University of Minnesota Press, 2012.

[15] D. M. Berry, Ed., *Understanding Digital Humanities.* New York: Palgrave Macmillan, 2012.

[16] P. Arthur and K. Bode, Eds., *Advancing Digital Humanities: Research, Methods, Theories.* Basingstoke, Hampshire: Palgrave Macmillan, 2014.

[17] A. Burdick, J. Drucker, P. Lunenfeld, T. Presner, and J. Schnapp, Eds., *Digital_Humanities.* Cambridge, MA, USA: MIT press, 2012.

[18] T. Bartscherer and R. Coover, *Switching Codes: Thinking Through Digital Technology in the Humanities and the Arts.* Chicago: University of Chicago Press, 2011.

[19] P. L. Shillingsburg, *From Gutenberg to Google: Electronic Representations of Literary Texts.* New York, NY, USA: Cambridge University Press, 2006.

[20] R. Busa and I. B. M. Corporation, *Sancti Thomae Aquinatis hymnorum ritualium varia specimina concordantiarum ...: a first example of word index automatically compiled and printed by IBM punched card machines*, ser. Archivum Philosophicum Aloisianum. Milano: Fratelli Bocca, Editori, 1951.

[21] R. Busa, "The annals of humanities computing: The index Thomisticus," *Computers and the Humanities*, vol. 14, no. 2, pp. 83–90, 1980. [Online]. Available: http://dx.doi.org/10.1007/BF02403798

[22] G. Stewart, G. Crane, and A. Babeu, "A new generation of textual corpora: Mining corpora from very large collections," in *Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL), Vancouver, BC, Canada.* New York, NY, USA: ACM, June 2007, pp. 356–365. [Online]. Available: http://doi.acm.org/10.1145/1255175.1255247

[23] P. Robinson, "The history of scholarly digital editions, plc," *Papers of the Bibliographical Society of Canada / Cahiers de la Société bibliographique du Canada*, vol. 51, no. 1, pp. 83–104, March 2013.

[24] G. Crane, D. Bamman, L. Cerrato, A. Jones, D. Mimno, A. Packel, D. Sculley, and G. Weaver, "Beyond Digital Incunabula: Modeling the next generation of Digital Libraries," in *Proceedings of the 10th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Alicante, Spain.* Berlin, Heidelberg: Springer-Verlag, September 2006, pp. 353–366. [Online]. Available: http://dx.doi.org/10.1007/11863878_30

[25] A. Bozzi, “Electronicpublishing and computational philology,” *Linguistica Computazionale*, vol. 24-25, no. A, pp. 63–86, 2006.

[26] V. A. Dearing, “Machine-assisted textual criticism,” *Computers and the Humanities*, vol. 4, no. 2, pp. 149–154, November 1969.

[27] G. R. Petty and W. M. Gibson, *Project Occult: the ordered computer collation of unprepared literary text.* New York, NY, USA: New York University Press, 1970.

[28] G. P. Zarri, “Some experiments on automated textual criticism,” *Bulletin Association for Literary and Linguistic Computing Stockport*, vol. 5, no. 3, pp. 266–290, 1977.

[29] G. Zarri, “Algorithms, Stemmata Codicum, and the Theories of Dom H. Quentin,” *The Computer and Literary Studies*, pp. 225–237, 1973.

[30] J. Froger, *La critique des textes et son automatisation.* Paris: Dunod, 1968, vol. 7.

[31] G. P. Verbrugghe, “Transliteration or transcription of Greek,” *The Classical World*, vol. 92, no. 6, pp. 499–511, July - August 1999.

[32] W. Ott, “Strategies and tools for textual scholarship: the Tübingen system of text processing programs (TUSTEP),” *Literary and Linguistic Computing*, vol. 15, no. 1, pp. 93–108, 2000. [Online]. Available: http://llc.oxfordjournals.org/content/15/1/93.abstract

[33] C. Thiele, “Tex and the Humanities,” *TUGboat*, vol. 17, no. 4, pp. 388 – 393, 1996.

[34] C. Beccari, “The teubner LATEX package: Typesetting classical Greek philology,” *TUGboat*, vol. 23, no. 3-4, pp. 276–282, 2002.

[35] A. Bozzi, “Digital documents and computational philology: the Digital Philology System (DiPhiloS),” *Informatica e Scienze Umane. Mezzo Secolo di Studi e Ricerche*, pp. 175–201, 2003.

[36] D. Schmidt, “Towards an interoperable digital scholarly edition,” *Journal of the Text Encoding Initiative*, no. 7, November 2014. [Online]. Available: http://jtei.revues.org/979

[37] F. Boschetti, A. M. Del Grosso, A. F. Khan, M. Lamé, and O. Nahli, "A top-down approach to the design of components for the philological domain," in *Book of abstract of Digital Humanities Conference (DH), Lausanne, Switzerland*. Alliance of Digital Humanities Organisations, July 2014, pp. 109–111.

[38] P. D'Iorio, "Nietzsche on new paths: The hypernietzsche project and open scholarship on the web," in *Friedrich Nietzsche, Edizioni e interpretazioni*, M. C. Fornari, Ed. Pisa: ETS, 2007, pp. 475–496.

[39] M. Langham and C. Ziegeler, *Cocoon: Building XML Applications.* Indianapolis: Pearson Education, 2002.

[40] (2003) Analytical System Tools and SGML/XML Integration Applications. Scholarly Digital Editions: Nottingham. [Online]. Available: http://sd-editions.com/anastasia/index.html

[41] G. Crane, B. Seales, and M. Terras, "Cyberinfrastructure for classical philology," *Digital Humanities Quarterly*, vol. 3, no. 1, 2009. [Online]. Available: http://www.digitalhumanities.org/dhq/vol/3/1/000023/000023.html

[42] D. Buzzetti, "Digital representation and the text model," *New Literary History*, vol. 33, no. 1, pp. 61–88, 2002. [Online]. Available: http://muse.jhu.edu/journals/new_literary_history/v033/33.1buzzetti.html

[43] F. Gibbs and T. Owens, "Building Better Digital Humanities Tools: Toward broader audiences and user-centered designs," *Digital Humanities Quarterly*, vol. 6, no. 2, 2012. [Online]. Available: http://www.digitalhumanities.org/dhq/vol/6/2/000136/000136.html

[44] A. Bozzi, "G2A: A Web application to study, annotate and scholarly edit ancient texts and their aligned translations," *Studia graeco-arabica*, vol. 3, pp. 159–171, 2013.

[45] G. Crane, B. Almas, A. Babeu, L. Cerrato, M. Harrington, D. Bamman, and H. Diakoff, "Student researchers, citizen scholars and the trillion word library," in *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL), Washington, DC, USA*. New York, NY, USA: ACM, June 2012, pp. 213–222. [Online]. Available: http://doi.acm.org/10.1145/2232817.2232857

[46] A. M. Del Grosso and O. Nahli, "Towards a flexible open-source software library for multi-layered scholarly textual studies: An Arabic case study dealing with semi-automatic language processing," in *Proceedings of 3rd IEEE International Colloquium, Information Science and Technology (CIST), Tetouan, Marocco.* Washington, DC, USA: IEEE, October 2014, pp. 285–290. [Online]. Available: http://dx.doi.org/10.1109/CIST.2014.7016633

[47] L. Burnard, "The Evolution of the Text Encoding Initiative: From Research Project to Research Infrastructure," *Journal of the Text Encoding Initiative*, no. 5, 2013. [Online]. Available: http://jtei.revues.org/811

[48] B. Bordalejo, "The texts we see and the works we imagine: The shift of focus of textual scholarship in the digitale age," in *Ecdotica*, G. M. Anselmi, E. Pasquini, and F. Rico, Eds. Roma: Carocci, 2013, vol. 10, pp. 64–75.

[49] G. Barabucci, A. Di Iorio, and F. Vitali, "Stemma codicum: analisi e generazione semi-automatica," *Quaderni DigiLab*, vol. 3, no. 1, pp. 129–145, 2014.

[50] E. Pierazzo, *Digital Scholarly Editing : Theories, Models and Methods.* Farnham Surrey: Ashgate, 2015.

[51] A. Teehan and J. G. Keating, "Appropriate use case modeling for humanities documents," *Literary and Linguistic Computing*, vol. 25, no. 4, pp. 381–391, 2010. [Online]. Available: http://llc.oxfordjournals.org/content/25/4/381.abstract

[52] T. Boudreau, J. Tulach, and R. Unger, "Decoupled Design: Building Applications on the NetBeans^{TM} Platform," in *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA), Portland, Oregon, USA.* New York, NY, USA: ACM, October 2006, pp. 631–631. [Online]. Available: http://doi.acm.org/10.1145/1176617.1176644

[53] P. M. Mell and T. Grance, "Sp 800-145. The NIST Definition of Cloud Computing," National Institute of Standards & Technology, Gaithersburg, MD, United States, Tech. Rep., 2011.

[54] R. Siemens and S. Schreibman, Eds., *A Companion to Digital Literary Studies*, ser. Blackwell Companions to Literature and Culture. Wiley Publishing, 2008.

[55] C. D. Manning, “Natural language processing for the Digital Humanities,” in *Workshop of Digital Humanities*, Stanford, June 2011. [Online]. Available: http://nlp.stanford.edu/~manning/courses/DigitalHumanities/

[56] A. Babeu, “Rome wasn’t digitized in a day: Building a cyberinfrastructure for digital classicists,” Council on Library and Information Resources, Tech. Rep., August 2011. [Online]. Available: http://www.clir.org/pubs/abstract/pub150abst.html

[57] M. Agosti and F. Tomasi, Eds., *Collaborative Research Practices and Shared Infrastructures for Humanities Computing*, Proceedings of revised papers of the 2nd Annual Conference of the Associazione per l’Informatica Umanistica e la Cultura Digitale (AIUCD), Padova, 11-12 December 2013. Padova: CLEUP, 2014.

[58] M. Terras, Ed., *Book of Abstracts of Digital Humanities Conference.* Joint International Conference of the Alliance of Digital Humanities Organizations, Lausanne, 7-21 July, 2014.

[59] S. Simske and S. Rönnau, Eds., *Proceedings of the ACM Symposium on Document Engineering (DocEng), Fort Collins, Colorado, USA, 16-19 September.* New York, NY, USA: ACM, 2014.

[60] A. Antonacopoulos and K. U. Schulz, Eds., *Proceedings of the 1st International Conference on Digital Access to Textual Cultural Heritage (DATeCH), Madrid, 19-20 May.* New York, NY, USA: ACM, 2014.

[61] P. M. Robinson, “Collate: A program for interactive collation of large textual traditions,” *Research in humanities computing*, vol. 3, pp. 32–45, 1994.

[62] D. Ribes and K. Baker, “Modes of social science engagement in community infrastructure design,” in *Communities and Technologies 2007*, C. Steinfield, B. Pentland, M. Ackerman, and N. Contractor, Eds. Springer London, 2007, pp. 107–130. [Online]. Available: http://dx.doi.org/10.1007/978-1-84628-905-7_6

[63] Q. Dombrowski, "What ever happened to project bamboo?" *Literary and Linguistic Computing*, vol. 29, no. 3, pp. 326–339, 2014. [Online]. Available: http://llc.oxfordjournals.org/content/29/3/326.abstract

[64] M. Hedges, H. Neuroth, K. M. Smith, T. Blanke, L. Romary, M. Küster, and M. Illingworth, "TextGrid, TEXTvre, and DARIAH: Sustainability of Infrastructures for Textual Scholarship," *Journal of the Text Encoding Initiative*, no. 5, June 2013. [Online]. Available: http://jtei.revues.org/774

[65] T. Blanke, M. Bryant, M. Hedges, A. Aschenbrenner, and M. Priddy, "Preparing DARIAH," in *Proceedings of 7th International Conference on E-Science (IEEE), Stockholm, Sweden*, December 2011, pp. 158–165.

[66] T. Váradi, S. Krauwer, P. Wittenburg, M. Wynne, and K. Koskenniemi, "Clarin: Common language resources and technology infrastructure," in *Proceedings of Language Resources and Evaluation Conference (LREC), Marrakech, Morocco*, N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, and D. Tapias, Eds. European Language Resources Association (ELRA), May 2008, pp. 1244–1248.

[67] R. Del Gratta, "Language resource infrastructure(s)," Ph.D. dissertation, Scuola di Dottorato in Ingegneria Leonardo da Vinci, Università di Pisa, 2011.

[68] C. Concordia, S. Gradmann, and S. Siebinga, "Not just another portal, not just another digital library: A portrait of Europeana as an application program interface," *IFLA Journal*, vol. 36, no. 1, pp. 61–69, 2010. [Online]. Available: http://ifl.sagepub.com/content/36/1/61.abstract

[69] V. Casarosa, C. Meghini, and S. Gardasevic, "Improving online access to archival data," in *Digital Libraries and Archives*, ser. Communications in Computer and Information Science, M. Agosti, F. Esposito, S. Ferilli, and N. Ferro, Eds. Springer Berlin Heidelberg, 2013, vol. 354, pp. 153–162. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35834-0_16

[70] C. Meghini, N. Aloia, and C. Concordia, "Designing and implementing a user-generated content service for Europeana," in *Proceedings of Information Technologies for Performing Arts, Media Access and*

Entertainment (ECHAP), Firenze. Firenze University Press, May 2012. [Online]. Available: http://dx.doi.org/10.1400/187218

[71] A. Jordanous, K. F. Lawrence, M. Hedges, and C. Tupman, "Exploring manuscripts: Sharing ancient wisdoms across the semantic web," in *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics (WIMS), Craiova, Romania.* New York, NY, USA: ACM, June 2012, pp. 44:1–44:12. [Online]. Available: http://doi.acm.org/10.1145/2254129.2254184

[72] M. Grassi, C. Morbidoni, M. Nucci, S. Fonda, and F. Piazza, "Pundit: augmenting web contents with semantics," *Literary and linguistic computing*, vol. 28, no. 4, pp. 640–659, 2013.

[73] A. Ciula, "The New Edition of the Letters of Vincent Van Gogh on the Web," *DHQ: Digital Humanities Quarterly*, vol. 4, no. 2, 2010. [Online]. Available: http://digitalhumanities.org/dhq/vol/4/2/000088/000088.html

[74] B. Nowviskie and J. McGann, "NINES: a federated model for integrating digital scholarship," Networked Infrastructure for Nineteenth-Century Electronic Scholarship, White paper, 2005. [Online]. Available: http://www.nines.org/about/wp-content/uploads/2011/12/9swhitepaper.pdf

[75] W. Ott, "Digital publishing: tools and products," *Poiesis and Praxis*, vol. 5, no. 2, pp. 81–112, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10202-007-0039-6

[76] W. Ott and T. Ott, "Critical editing with TXSTEP," in *Book of Abstracts of the Digital Humanities conference, Lausanne, Switzerland*, M. Terras, Ed. Alliance of Digital Humanities Organisations, July 2014, pp. 509–513.

[77] P. Mascellani and P. D. Napoletani, "MauroTeX - A language for electronic critical editions," in *Proceedings of International Cultural Heritage Informatics Meeting (ICHIM), Milan, Italy*, D. Bearman and F. Garzotto, Eds., vol. 2. Pittsburgh, PA, USA: Archives and Museum Informatics, September 2001, pp. 223–241.

[78] M. Dominici and P. D. Napolitani, "Edizione con LaTeX delle opere di Francesco Maurolico," *ArsTEXnica*, pp. 75–82, October 2006.

[79] P. Heckel, “A technique for isolating differences between files,” *Communications of the ACM*, vol. 21, no. 4, pp. 264–268, April 1978. [Online]. Available: http://doi.acm.org/10.1145/359460.359467

[80] R. Haentjens Dekker, D. van Hulle, G. Middell, V. Neyt, and J. van Zundert, “Computer-supported collation of modern manuscripts: CollateX and the Beckett digital manuscript project,” *Literary and Linguistic Computing*, 2014. [Online]. Available: http://dx.doi.org/10.1093/llc/fqu007

[81] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022283670900574

[82] J. Bourdaillet and J.-G. Ganascia, “Practical block sequence alignment with moves,” in *Proceedings of 1st International on Language and Automata Theory and Applications (LATA), Tarragona, Spain*, March-April 2007, pp. 199–210.

[83] W. Meier, “eXist: An open source native XML database,” in *Web, Web-Services, and Database Systems*, ser. Lecture Notes in Computer Science, A. Chaudhri, M. Jeckle, E. Rahm, and R. Unland, Eds. Springer Berlin Heidelberg, 2003, vol. 2593, pp. 169–183. [Online]. Available: http://dx.doi.org/10.1007/3540365605_13

[84] A. H. Renear, E. Mylonas, and D. Durand, “Refining our notion of what text really is: The problem of overlapping hierarchies,” *Research in Humanities Computing*, vol. 4, pp. 263–280, 1996.

[85] A. Di Iorio, S. Peroni, and F. Vitali, “Towards markup support for full GODDAGs and beyond: the EARMARK approach,” in *Proceedings of balisage: The markup conference, Montréal, Canada*, ser. Balisage Series on Markup Technologies, vol. 3, August 2009. [Online]. Available: http://www.balisage.net/Proceedings/vol3/html/Peroni01/BalisageVol3-Peroni01.html

[86] D. Schmidt, “The inadequacy of embedded markup for cultural heritage texts,” *Literary and Linguistic Computing*, vol. 25, no. 3, pp. 337–356, 2010. [Online]. Available: http://llc.oxfordjournals.org/content/25/3/337.abstract

[87] D. Schmidt and R. Colomb, "A data structure for representing multi-version texts online," *International Journal of Human-Computer Studies*, vol. 67, no. 6, pp. 497–514, June 2009. [Online]. Available: http://dx.doi.org/10.1016/j.ijhcs.2009.02.001

[88] M. Crochemore, C. Hancart, and T. Lecroq, *Algorithms on Strings*. New York, NY, USA: Cambridge University Press, 2007.

[89] J. D. Allen, D. Anderson, J. Becker, R. Cook, M. Davis, P. Edberg, M. Everson, A. Freytag, L. Iancu, R. Ishida, J. H. Jenkins, K. Lunde, R. McGowan, L. Moore, E. Muller, A. Phillips, R. Pournader, M. Suignard, and K. Whistler, Eds., *The Unicode Standard, Version 7.0*. Boston MA, USA: Addison Wesley Longman Publishing Co., Inc., October 2014, vol. 7.0.

[90] J. M. Aliprand, "Unicode and ISO/IEC 10646: An overview," *McCallum and Ertel [ME94]*, pp. 87–102, 1993.

[91] S. Ferilli, *Automatic Digital Document Processing and Management: Problems, Algorithms and Techniques*. Berlin Heidelberg: Springer, 2011.

[92] L. Burnard, K. O'Brien O'Keeffe, and J. Unsworth, "Editors' introduction," in *Electronic textual editing*, L. Burnard, K. O'Brien O'Keeffe, and J. Unsworth, Eds. New York, NY, USA: Modern Language Association of America, 2006, pp. 11–12.

[93] F. Tomasi, "L'edizione digitale e la rappresentazione della conoscenza. Un esempio: Vespasiano da Bisticci e le sue lettere," *Ecdotica*, vol. 9, pp. 264–286, 2012.

[94] L. Burnard. (2014) TEI P5: Guidelines for Electronic Text Encoding and Interchange. Version 2.7.0. [Online]. Available: http://www.tei-c.org/Vault/P5/2.7.0/doc/tei-p5-doc/en/html/index.html

[95] J. A. Rydberg-Cox, *Digital Libraries and the Challenges of Digital Humanities*, R. Rikowski, Ed. Oxford: Chandos Publishing, 2006.

[96] S. Peroni, F. Poggi, and F. Vitali, "Overlapproaches in documents: a definitive classification (in OWL, 2!)," in *Proceedings of Balisage: The Markup Conference, Washington, DC*, ser. Balisage Series on Markup Technologies, vol. 13, August 2014. [Online]. Available: http://www.balisage.net/Proceedings/vol13/html/Peroni01/BalisageVol13-Peroni01.html

[97] A. Vohra and D. Vohra, *Pro XML Development with Java Technology (Pro)*. Berkely, CA, USA: Apress, 2006.

[98] D. Schmidt, "Merging multi-version texts: a general solution to the overlap problem," in *Proceeding of Balisage: The Markup Conference, Montréal, Canada*, ser. Belisage Series on Markup Technologies, vol. 3, August 2009, pp. 497–514. [Online]. Available: http://www.balisage.net/Proceedings/vol3/html/Schmidt01/BalisageVol3-Schmidt01.html

[99] N. Guarino, D. Oberle, and S. Staab, "What is an ontology?" in *Handbook on Ontologies*, ser. International Handbooks on Information Systems, S. Staab and R. Studer, Eds. Springer Berlin Heidelberg, 2009, pp. 1–17. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92673-3_0

[100] M. Doerr, S. Gradmann, S. Hennicke, A. Isaac, C. Meghini, and H. van de Sompel, "The Europeana Data Model (EDM)," in *World Library and Information Congress: 76th IFLA General Conference and Assembly, Gothenburg, Sweden*, August 2010, pp. 10–12.

[101] A. Gerber and J. Hunter, "A compound object authoring and publishing tool for literary scholars based on the IFLA-FRBR," *International Journal of Digital Curation*, vol. 4, no. 2, pp. 28–42, 2009.

[102] D. Broeder, M. Kemps-Snijders, D. van Uytvanck, M. Windhouwer, P. Withers, P. Wittenburg, and C. Zinn, "A data category registry- and component-based metadata framework," in *Proceedings of the 7th conference on International Language Resources and Evaluation (LREC), Malta*, N. Calzolari, B. Maegaard, J. Mariani, J. Odjik, K. Choukri, S. Piperidis *et al.*, Eds. European Language Resources Association (ELRA), May 2010, pp. 43–47.

[103] A. Maslov, J. Creel, A. Mikeal, S. Phillips, J. Leggett, and M. McFarland, "Adding OAI–ORE support to repository platforms," *Journal of Digital Information*, vol. 11, no. 1, 2010. [Online]. Available: https://journals.tdl.org/jodi/index.php/jodi/article/view/749

[104] N. Smith, "Citation in classical studies," *Digital Humanities Quarterly*, vol. 3, no. 1, pp. 1–10, 2009.

[105] J. Tiepmar, C. Teichmann, G. Heyer, M. Berti, and G. Crane, "A new implementation for Canonical Text Services," in *Proceedings of the 8th Workshop*

on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH), Gothenburg, Sweden, K. Zervanou and C. Vertan, Eds., European Chapter Association for Computational Linguistics (EACL). New York, NY, USA: ACL, April 2014, pp. 1–8.

[106] D. N. Smith and C. W. Blackwell, "Four URLs, Limitless Apps: separation of concerns in the Homer Multitext architecture," in *Donum natalicium digitaliter confectum Gregorio Nagy septuagenario a discipulis collegis familiaribus oblatum (A Virtual Birthday Gift Presented to Gregory Nagy on Turning Seventy by his Students, Colleagues, and Friends) édition*, L. Muellner, Ed. The Center of Hellenic Studies of Harvard University, 2012. [Online]. Available: http://chs.harvard.edu/CHS/article/display/4846

[107] G. Crane, B. Almas, A. Babeu, L. Cerrato, A. Krohn, F. Baumgart, M. Berti, G. Franzini, and S. Stoyanova, "Cataloging for a Billion Word Library of Greek and Latin," in *Proceedings of the 1st International Conference on Digital Access to Textual Cultural Heritage (DATeCH), Madrid, Spain.* New York, NY, USA: ACM, May 2014, pp. 83–88. [Online]. Available: http://doi.acm.org/10.1145/2595188.2595190

[108] J. Bradley and P. Vetch, "Supporting annotation as a scholarly tool – Experiences from the online Chopin Variorum Edition," *Literary and Linguistic Computing*, vol. 22, no. 2, pp. 225–241, 2007. [Online]. Available: http://llc.oxfordjournals.org/content/22/2/225.abstract

[109] M. Agosti and N. Ferro, "A formal model of annotations of digital content," *ACM Transactions on Information Systems (TOIS)*, vol. 26, no. 1, November 2007. [Online]. Available: http://doi.acm.org/10.1145/1292591.1292594

[110] C. Mattmann and J. Zitting, *Tika in Action.* Greenwich, CT, USA: Manning Publications Co., 2011.

[111] M. McCandless, E. Hatcher, and O. Gospodnetić, *Lucene in action; 2nd ed.* Stamford Conn: Manning Pub, 2010.

[112] G. Wilcock, *Introduction to Linguistic Annotation and Text Analytics*, 1st ed. Morgan & Claypool Publishers, 2009.

[113] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD*

Explorations Newsletter, vol. 11, no. 1, pp. 10–18, November 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[114] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in Action.* Greenwich, CT, USA: Manning Publications Co., 2011.

[115] C. Carpineto, S. Osiński, G. Romano, and D. Weiss, "A survey of web clustering engines," *ACM Computing Surveys*, vol. 41, no. 3, pp. 17:1–17:38, July 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541884

[116] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Baltimore, Maryland USA*. Stroudsburg, PA, USA: Association for Computational Linguistics, June 2014, pp. 55–60.

[117] A. Taylor, M. Marcus, and B. Santorini, "The Penn Treebank: An overview," in *Treebanks*, ser. Text, Speech and Language Technology, A. Abeillé, Ed. Springer Netherlands, 2003, vol. 20, pp. 5–22. [Online]. Available: http://dx.doi.org/10.1007/978-94-010-0201-1_1

[118] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL), Edmonton, Canada*, vol. 1. Stroudsburg, PA, USA: ACL, May-June 2003, pp. 173–180. [Online]. Available: http://dx.doi.org/10.3115/1073445.1073478

[119] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by Gibbs sampling," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL), Ann Arbor, Michigan.* Stroudsburg, PA, USA: Association for Computational Linguistics, June 2005, pp. 363–370. [Online]. Available: http://dx.doi.org/10.3115/1219840.1219885

[120] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic *et al.*, “Developing language processing components with GATE version 7 (a user guide),” The University of Sheffield, Department of Computer Science, Sheffield, Tech. Rep., November 2012.

[121] T. Götz and O. Suhre, “Design and implementation of the UIMA common analysis system,” *IBM System Journal*, vol. 43, no. 3, pp. 476–489, July 2004. [Online]. Available: http://dx.doi.org/10.1147/sj.433.0476

[122] E. Pierazzo, “A rationale of digital documentary editions,” *Literary and Linguistic Computing*, vol. 26, no. 4, pp. 463–477, 2011. [Online]. Available: http://llc.oxfordjournals.org/content/26/4/463.abstract

[123] M. Crochemore and W. Rytter, *Jewels of stringology.* Singapore, River Edge, NJ: World Scientific, 2003.

[124] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* New York, NY, USA: Cambridge University Press, 1997.

[125] J.-B. Camps and F. Cafiero, “Genealogical variant locations and simplified stemma: a test case,” in *Analysis of Ancient and Medieval Texts and Manuscripts: Digital Approaches*, C. M. T. L Andrews, Ed. Leuven: Brepols Publishers, 2012, ch. 3, pp. 71–97.

[126] F. Boschetti, “Alignment of variant readings for linkage of multiple annotations,” in *In Proceedings of Electronic Corpora of Ancient Languages 2007, Prague, Czech Republic*, P. Zemanek, Ed., July 2008, pp. 11–24.

[127] D. Sankoff and J. B. Kruskal, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Center for the Study of Language and Information, December 1999.

[128] J. Holub, “Editorial: Stringology algorithms,” *Discrete Applied Mathematics*, vol. 163, no. 3, pp. 237–238, January 2014. [Online]. Available: http://dx.doi.org/10.1016/j.dam.2013.11.009

[129] H.-J. Boeckenhauer and D. Bongartz, *Algorithmic Aspects of Bioinformatics*, ser. Natural Computing. Springer Berlin Heidelberg, 2007. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71913-7_1

[130] U. Manber and G. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM Journal Computing*, vol. 22, no. 5, pp. 935–948, October 1993. [Online]. Available: http://dx.doi.org/10.1137/0222058

[131] P. Husemann and J. Stoye, "Phylogenetic comparative assembly," in *Proceedings of the 9th International Conference on Algorithms in Bioinformatics (WABI), Philadelphia, PA, USA*. Berlin, Heidelberg: Springer-Verlag, September 2009, pp. 145–156. [Online]. Available: http://dl.acm.org/citation.cfm?id=1812906.1812919

[132] R. C. Edgar and S. Batzoglou, "Multiple sequence alignment," *Current Opinion in Structural Biology*, vol. 16, no. 3, pp. 368–373, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959440X06000704

[133] M. Spencer and C. Howe, "Collating texts using progressive multiple alignment," *Computers and the Humanities*, vol. 38, no. 3, pp. 253–270, 2004. [Online]. Available: http://dx.doi.org/10.1007/s10579-004-8682-1

[134] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022283681900875

[135] D.-F. Feng and R. F. Doolittle, "Progressive sequence alignment as a prerequisite to correct phylogenetic trees," *Journal of Molecular Evolution*, vol. 25, no. 4, pp. 351–360, 1987. [Online]. Available: http://dx.doi.org/10.1007/BF02603120

[136] P. Robinson, "The concept of the work in the digital age," in *Ecdotica*, G. M. Anselmi, E. Pasquini, and F. Rico, Eds. Roma: Carocci, 2013, vol. 10, pp. 13–41.

[137] P. V. Baret, P. Robinson, and C. Macé, "Testing methods on an artificially created textual tradition," in *The evolution of texts: confronting stemmatological and genetical methods*, C. Macé, P. Baret, A. Bozzi, and L. Cignoni, Eds., vol. 24-25, Proceedings of the international workshop held in Louvain-la-Neuve on September 1-2, 2004. Pisa, Roma: Istituti

editoriali e poligrafici internazionali, 2006, pp. 255–283. [Online]. Available: http://digital.casalini.it/an/2212910

[138] M. C. Passarotti, "Towards textual drift modelling in computational philology," in *Linguistica Computazionale*, C. Macé, P. Baret, A. Bozzi, and L. Cignoni, Eds., vol. 24-25, Proceedings of the international workshop held in Louvain-la-Neuve on September 1-2, 2004. Pisa-Rome: Istituti Editoriali e Poligrafici Internazionali, 2006, pp. 63–86.

[139] H. Bast and M. Celikik, "Efficient fuzzy search in large text collections," *ACM Transactions on Information Systems*, vol. 31, no. 2, pp. 10:1–10:59, May 2013. [Online]. Available: http://doi.acm.org/10.1145/2457465.2457470

[140] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, November 1975. [Online]. Available: http://doi.acm.org/10.1145/361219.361220

[141] F. P. Miller, A. F. Vandome, and J. McBrewster, *Levenshtein Distance*. Alpha Press, 2009.

[142] Z. S. Harris, "Distributional structure," *Word*, 1954.

[143] A. Lenci and G. Benotto, "Identifying Hypernyms in Distributional Semantic Spaces," in *Proceedings of the 1st Joint Conference on Lexical and Computational Semantics: Proceedings of the Main Conference and the Shared Task: Proceedings of the 6th International Workshop on Semantic Evaluation (SemEval), Montréal, Canada*, vol. 1,2. Stroudsburg, PA, USA: Association for Computational Linguistics, June 2012, pp. 75–79. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387636.2387650

[144] R. Greenlaw and S. Kantabutra, "Survey of clustering: Algorithms and applications," *International Journal of Information Retrieval Research*, vol. 3, no. 2, pp. 1–29, April 2013. [Online]. Available: http://dx.doi.org/10.4018/ijirr.2013040101

[145] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[146] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, 1st ed. Cambridge, MA, USA: The MIT Press, June 1999.

[147] A. Tonazzini, E. Salerno, V. Palleschi, G. Bianco, and F. De Filippo, "Extracting information from multimodal images of documents and artworks," in *Proceedings of 6th International Congress on Science and Technology for the Safeguard of Cultural Heritage in the Mediterranean Basin, Athens, Greece*, vol. 3. Rome: Valmar, October 2013, pp. 196–204.

[148] A. Bozzi, M. M. Morales, and M. Rufino, "Imago et umbra programma di digitalizzazione per l'archivio storico della Pontificia Università Gregoriana: criteri, metodi e strumenti," *Digitalia*, vol. 5, no. 2, pp. 79–99, 2010.

[149] A. Prescot, "The electronic Beowulf and digital restoration," *Literary and linguistic computing*, vol. 12, no. 3, pp. 186–195, 1997.

[150] K. S. Kiernan, "Digital image processing and the Beowulf manuscript," *Literary and Linguistic Computing*, vol. 6, no. 1, pp. 20–27, 1991. [Online]. Available: http://llc.oxfordjournals.org/content/6/1/20.abstract

[151] R. Scopigno, "Visual media for cultural heritage: an opportunity for assessing, finding limitations and enhancing technologies," in *Proceedings of 18th Central European Seminar on Computer Graphics, Smolenice, Slovakia*, M. I. Michael Wimmer, Jiri Hladuvka, Ed., vol. 1. Vienna University of Technology, May 2014, pp. 5–6.

[152] S. Marinai and H. Fujisawa, *Machine Learning in Document Analysis and Recognition*, 1st ed. Berlin: Springer Publishing Company, Incorporated, 2010.

[153] F. Ciotti, M. Daquino, and F. Tomasi, "Text encoding initiative semantic modeling. A conceptual workflow proposal," in *Proceedings of 11th Italian Research Conference on Digital Libraries (IRCDL), Bolzano, Italy*, January 2015. [Online]. Available: http://ircdl2015.unibz.it/papers/paper-10.pdf

[154] F. Boschetti, A. Cimino, F. Dell'Orletta, G. E. Lebani, L. Passaro, P. Picchi, G. Venturi, S. Montemagni, and A. Lenci, "Computational analysis of historical documents: An application to italian war bulletins in world war I and II," in *Proceedings of the LREC Workshop on Language Resources and*

Technologies for Processing and Linking Historical Documents and Archives. Deploying Linked Open Data in Cultural Heritage, (LRT4HDA), Reykjavik, May 2014, pp. 70–79.

[155] F. Ciotti and A. Ciula, Eds., *The Linked TEI: Text Encoding in the Web.* TEI Conference, DIGILAB, Rome, Italy 2-5 October, 2013.

[156] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, ser. Synthesis Lectures on the Semantic Web: Theory and Technology. San Rafael, California, USA: Morgan & Claypool, 2011, vol. 1, no. 1. [Online]. Available: http://dx.doi.org/10.2200/S00334ED1V01Y201102WBE001

[157] A. E. C. Basave, G. Rizzo, A. Varga, M. Rowe, M. Stankovic, and A. S. Dadzie, "Making sense of Microposts (#Microposts2014) named entity extraction and linking challenge," in *Proceedings of the 4th Workshop on Making Sense of Microposts co-located with the 23rd International World Wide Web Conference, Seoul, Korea*, April 2014, pp. 54–60.

[158] M. Passarotti, "LEMLAT. Uno strumento per la lemmatizzazione morfologica automatica del latino," *Journal of Latin Linguistics*, vol. 9, no. 3, pp. 107–128, 2007.

[159] B. McGillivray, *Methods in Latin Computational Linguistics.* Leiden: Brill, 2013.

[160] F. Sebastiani, "Text categorization," in *Text Mining and its Applications*, A. Zanasi, Ed. WIT Press, 2005. [Online]. Available: http://www.isti.cnr.it/People/F.Sebastiani/Publications/TM05.pdf

[161] P. Ferragina, R. González, G. Navarro, and R. Venturini, "Compressed text indexes: From theory to practice," *Journal of Experimental Algorithmics (JEA)*, vol. 13, pp. 12:1.12–12:1.31, February 2009. [Online]. Available: http://doi.acm.org/10.1145/1412228.1455268

[162] M. Ciaramita, G. Attardi, F. Dell'Orletta, and M. Surdeanu, "DeSRL: A linear-time semantic role labeling system," in *Proceedings of the 12th Conference on Computational Natural Language Learning (CoNLL), Manchester, United Kingdom.* Stroudsburg, PA, USA: Association for

Computational Linguistics, August 2008, pp. 258–262. [Online]. Available: http://dl.acm.org/citation.cfm?id=1596324.1596371

[163] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[164] M. Abrate, A. M. Del Grosso, E. Giovannetti, A. L. Duca, D. Luzzi, L. Mancini, A. Marchetti, I. Pedretti, and S. Piccini, "Sharing cultural heritage: the Clavius on the Web project," in *Proceedings of the 9th international conference on Language Resources and Evaluation (LREC), Reykjavik*, N. Calzolari, K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, and S. Piperidis, Eds. European Language Resources Association (ELRA), May 2014, pp. 627–634.

[165] P. Halácsy, A. Kornai, and C. Oravecz, "HunPos: An open source trigram tagger," in *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics on Interactive Poster and Demonstration Sessions (ACL), Prague, Czech Republic*. Stroudsburg, PA, USA: ACL, June 2007, pp. 209–212. [Online]. Available: http://dl.acm.org/citation.cfm?id=1557769.1557830

[166] D. Bamman and G. Crane, "The ancient Greek and Latin dependency Treebanks," in *Language Technology for Cultural Heritage*, ser. Theory and Applications of Natural Language Processing, C. Sporleder, A. van den Bosch, and K. Zervanou, Eds. Springer Berlin Heidelberg, 2011, pp. 79–98. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20227-8_5

[167] H. Beigi, "Hidden Markov Modeling (HMM)," in *Fundamentals of Speaker Recognition*. Springer US, 2011, pp. 411–463. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-77592-0_13

[168] S. Benedetto and E. Biglieri, *Viterbi algorithm*, ser. Information Technology: Transmission, Processing, and Storage. Springer US, 2002, ch. F, pp. 807–815. [Online]. Available: http://dx.doi.org/10.1007/0-306-46961-8_20

[169] F. de Jong, "NLP and the humanities: The revival of an old liaison," in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Athens, Greece*.

Stroudsburg, PA, USA: ACL, March-April 2009, pp. 10–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1609067.1609168

[170] E. Foster, "Introduction to software engineering," in *Software Engineering.* Apress, 2014, pp. 3–20. [Online]. Available: http://dx.doi.org/10.1007/978-1-4842-0847-2_1

[171] M. Fowler, "The state of design," *IEEE Software*, vol. 22, no. 6, pp. 12–13, 16, November 2005. [Online]. Available: http://dx.doi.org/10.1109/MS.2005.166

[172] R. S. Pressman, *Software engineering: a practitioner's approach.* Palgrave Macmillan, 2005.

[173] W. McCarty, "Signs of times present and future," *Human Discussion Group*, vol. 22, no. 218, 2008.

[174] A. M. Del Grosso and F. Boschetti, "Collaborative multimedia platform for computational philology, CoPhi architecture," in *Proceedings of the 5th International Conferences on Advances in Multimedia (MMEDIA), Venice, Italy*, P. Davis, Ed., International Academy, Research, and Industry Association. IARIA, April 2013, pp. 46–51.

[175] F. Boschetti, A. M. Del Grosso, and M. Lamé, "Data sets and software components: Adjustment and reuse," in *Proceedings of the Papyrus and the Hypertext. Athenaeus in the Scholarly Kitchen, Paris, France.* Université Paris-Ouest and ANHIMA, May 2012, selected paper.

[176] Object Management Group, *Unified Modeling Language Specification*, 2000.

[177] G. Booch, *Software Component with ADA: Structures, Tools, and Subsystems*, 1st ed., ser. The Benjamin/Cummings in Ada and Software Engineering. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., April 1987.

[178] G. T. Heineman and W. T. Councill, Eds., *Component-based Software Engineering: Putting the Pieces Together.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[179] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[180] A. Bozzi and A. M. Del Grosso, “Progettazione, sviluppo e gestione di una infrastruttura filologico-computazionale per la produzione, interrogazione e pubblicazione sul web di documenti digitali,” in *Percorsi migranti: uomini, diritti, lavoro, linguaggi*, G. C. Bruno, I. Caruso, M. Sanna, and I. Vellecco, Eds. Milano: Mc Graw Hill, 2011, pp. 339–369.

[181] S. Ashmore and K. Runyan, *Introduction to Agile Methods.* Upper Saddle River, NJ: Addison-Wesley Professional, Pearson Education, 2014.

[182] H. Beyer, *User-centered Agile Methods*, ser. Synthesis lectures on human-centered informatics. San Rafael, California: Morgan & Claypool, 2010.

[183] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, “An examination of software engineering work practices,” in *Proceedings of the 1st Decade High Impact Papers of Centers for Advanced Studies Conference (CASCON), Toronto, Ontario, Canada.* Riverton, NJ, USA: IBM Corporation, November 2010, pp. 174–188. [Online]. Available: http://dx.doi.org/10.1145/1925805.1925815

[184] D. Rosenberg and M. Stephens, *Use Case Driven Object Modeling with UML. Theory and Practice*, 2nd ed. Berkely, CA, USA: Apress, 2013.

[185] M. Collins-Cope, D. Rosenberg, and M. Stephens, *Agile Development with ICONIX Process: People, Process, and Pragmatism.* Berkely, CA, USA: Apress, 2005.

[186] G. Booch, J. Conallen, B. J. Young, K. A. Houston, R. A. Maksimchuk, and M. W. Engle, *Object-Oriented Analysis and Design with Applications*, 3rd ed. Addison-Wesley, April 2007.

[187] S. Meyers, “The most important design guideline? [user interface],” *IEEE Software*, vol. 21, no. 4, pp. 14–16, July 2004. [Online]. Available: http://dx.doi.org/10.1109/MS.2004.29

[188] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[189] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual*, 2nd ed. Addison-Wesley Professional, 2010.

[190] J. Bloch, "How to design a good API and why it matters," in *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA), Portland, Oregon, USA*. New York, NY, USA: ACM, October 2006, pp. 506–507. [Online]. Available: http://doi.acm.org/10.1145/1176617.1176622

[191] J. Blanchette, *The Little Manual of API Design*, Trolltech a Nokia company, June 2008.

[192] M. Cohn, *User Stories Applied: For Agile Software Development*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[193] F. Boschetti, R. Del Gratta, A. M. Del Grosso, M. Monachini, and O. Nahli, "Collaborative philology on the way to Web services: the case of CoPhi-Wordnet," in *Proceedings of the 2nd International Workshop on Worldwide Language Service Infrastructure (WLSI), Kyoto, Japan*, Y. Murakami and D. Lin, Eds., January 2015, selected paper.

[194] F. Boschetti, A. Bozzi, and A. M. Del Grosso, "Library of components for the computational philological domain dealing with TEI markup guidelines: CoPhiLib," in *Book of Abstracts of the TEI Conference and Members Meeting. The Linked TEI: Text Encoding in the Web, Rome, Italy*, F. Ciotti and A. Ciula, Eds. Rome: Universitalia, October 2013, pp. 160–162.

[195] M. Henning, "API design matters," *Communications of the ACM*, vol. 52, no. 5, pp. 46–56, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1506409.1506424

[196] M. Lamé, V. Valchera, and F. Boschetti, "Epigrafia digitale. Paradigmi di rappresentazione per il trattamento digitale delle epigrafi," *Epigraphica: periodico internazionale di epigrafia*, vol. 1-2, no. 74, pp. 331–338, 2012.

[197] L. Benedetti, F. Boschetti, A. M. Del Grosso, and M. Lamé, "La matière épigraphique dans un espace numérique: l'importance du support archéologique," in *Proceedings of Journées d'Informatique et Archéologie de Paris (JIAP), Paris, France.* Université de Paris 1 Panthéon Sorbonne (UFR 03), CNRS UMR 7041 Arscan, June 2012, selected paper.

[198] F. Soler, J. C. Torres, A. J. León, and M. V. Luzón, "Design of cultural heritage information systems based on information layers," *Journal on Computing and Cultural Heritage (JOCCH)*, vol. 6, no. 4, pp. 15:1–15:17, December 2013. [Online]. Available: http://doi.acm.org/10.1145/2532630.2532631

[199] B. Almas and M.-C. Beaulieu, "Developing a new integrated editing platform for source documents in classics," *Literary and Linguistic Computing*, vol. 28, no. 4, pp. 493–503, 2013. [Online]. Available: http://llc.oxfordjournals.org/content/28/4/493.abstract

[200] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[201] R. Daigneau, *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*, 1st ed. Upper Saddle River, NJ: Addison-Wesley Professional, 2011.

[202] J. Tulach, *Practical API Design: Confessions of a Java Framework Architect*, 1st ed. Berkely, CA, USA: Apress, 2008.

[203] D. Buzzetti, "Digital editions and text processing," in *Text editing, print and the digital world*, ser. Digital Research in the Arts and Humanities, K. S. Marilyn Deegan, Ed. Farnham Surrey: Ashgate, 2009, pp. 45–62.

[204] F. Boschetti, "A corpus-based approach to philological issues," Ph.D. dissertation, University of Trento, Trento, March 2010.

[205] M. Reddy, *API Design for C++*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[206] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture, On Patterns and Pattern Languages*, ser. Pattern-Oriented Software Architecture. Hoboken: John Wiley & Sons, 2007.

[207] B. Liskov, "Keynote address - data abstraction and hierarchy," *SIGPLAN Notices, Special issue: OOPSLA*, vol. 23, no. 5, pp. 17–34, January 1987. [Online]. Available: http://doi.acm.org/10.1145/62139.62141

[208] R. C. Martin, “Design principles and design patterns,” *Object Mentor*, no. 1, pp. 1–34, 2000.

[209] J. Bloch, *Effective Java*, 2nd ed., ser. Java Series. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.

[210] F. Boschetti, “Methods to extend Greek and Latin corpora with variants and conjectures: Mapping critical apparatuses onto reference text,” in *Proceedings of the Corpus Linguistics Conference (CL), Birmingham, UK*, M. Davies, P. Rayson, S. Hunston, and P. Danielsson, Eds. University of Birmingham, July 2007, pp. 150:1–150:11.

[211] G. Genette, *Paratexts: Thresholds of Interpretation*, ser. Literature, Culture, Theory. Cambridge: Cambridge University Press, 1997.

[212] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[213] S. Fraser, E. Gamma, R. Helm, and R. Johnson, “Design patterns: Beginnings and futures,” in *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA), Portland, Oregon, USA*. New York, NY, USA: ACM, October 2006, pp. 934–934. [Online]. Available: http://doi.acm.org/10.1145/1176617.1176748

[214] S. Montemagni, “DH@ILC,” in *Collaborative Research Practices and Shared Infrastructures for Humanities Computing. Proceedings of revised papers of the 2nd Annual Conference of the Associazione per l’Informatica Umanistica e la Cultura Digitale (AIUCD), Padova*, M. Agosti and F. Tomasi, Eds. CLEUP, December 2013, pp. 101–114.

[215] K. Arnold, “Programmers are people, too,” *Queue*, vol. 3, no. 5, pp. 54–59, June 2005. [Online]. Available: http://doi.acm.org/10.1145/1071713.1071731

[216] D. L. Parnas, P. C. Clements, and D. M. Weiss, “The modular structure of complex systems,” *IEEE Transaction on Software Engineering*, vol. 11, no. 3, pp. 259–266, March 1985. [Online]. Available: http://dx.doi.org/10.1109/TSE.1985.232209

[217] K. Leino, M. Rustan, and G. Nelson, “Data abstraction and information hiding,” *ACM Transactions on Programming Languages and Systems*, vol. 24, no. 5, pp. 491–553, September 2002. [Online]. Available: http://doi.acm.org/10.1145/570886.570888

[218] D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, December 1972. [Online]. Available: http://doi.acm.org/10.1145/361598.361623

[219] M. P. Robillard, “What makes APIs hard to learn? Answers from developers,” *IEEE Software*, vol. 26, no. 6, pp. 27–34, November 2009. [Online]. Available: http://dx.doi.org/10.1109/MS.2009.193

[220] J. Stylos, A. Faulring, Z. Yang, and B. Myers, “Improving API documentation using API usage information,” in *Visual Languages and Human-Centric Computing (VL/HCC), Corcallis, Oregon*, ser. IEEE Symposium on, September 2009, pp. 119–126.

[221] T. Grill, O. Polacek, and M. Tscheligi, “Methods towards API usability: A structural analysis of usability problem categories,” in *Proceedings of the 4th International Conference on Human-Centered Software Engineering, Toulouse, France.* Berlin, Heidelberg: Springer-Verlag, October 2012, pp. 164–180. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34347-6_10

[222] B. A. Myers, S. Y. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, “Studying the documentation of an API for enterprise service-oriented architecture,” *J. Organ. End User Comput.*, vol. 22, no. 1, pp. 23–51, January 2010. [Online]. Available: http://dx.doi.org/10.4018/joeuc.2010101903

[223] M. A. Khan, S. Muhammad, and T. Muhammad, “Identifying performance issues based on method invocation patterns of an API,” in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE), London, England, United Kingdom.* New York, NY, USA: ACM, May 2014, pp. 51:1–51:6. [Online]. Available: http://doi.acm.org/10.1145/2601248.2601277

[224] C. De Roover, R. Lammel, and E. Pek, "Multi-dimensional exploration of API usage," in *Proceedings of the 21st IEEE International Conference on Program Comprehension (ICPC), San Francisco, CA, USA*. Washington, DC, USA: IEEE, May 2013, pp. 152–161.

[225] D. Dig and R. Johnson, "The role of refactorings in API evolution," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM), Budapest, Hungary.* Washington, DC, USA: IEEE, September 2005, pp. 389–398.

[226] G. Rooney, "Preserving backward compatibility," FactSet Research Systems, Open Paper on O'Reilly OnLamp.com, 2005. [Online]. Available: http://www.onlamp.com/pub/a/onlamp/2005/02/17/backwardscompatibility.html

[227] H. Boeck, *The Definitive Guide to NetBeans^{TM} Platform*, 1st ed. Apress, May 2009.

[228] M. Naftalin and P. Wadler, *Java Generics and Collections.* O'Reilly Media, Inc., 2006.

[229] C. Parnin, C. Bird, and E. Murphy-Hill, "Adoption and use of Java generics," *Empirical Software Engineering*, vol. 18, no. 6, pp. 1047–1089, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10664-012-9236-6

[230] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture: A System of Patterns*, ser. Pattern-oriented software architecture. Wiley India Pvt. Limited, 2008, vol. 1.

[231] P. Mastandrea and L. Tessarolo, "Da Musisque Deoque a Memorata Poetis. Le vie della ricerca intertestuale," in *Collaborative Research Practices and Shared Infrastructures for Humanities Computing. Proceedings of revised papers of the 2nd Annual Conference of the Associazione per l'Informatica Umanistica e la Cultura Digitale (AIUCD), Padova*, M. Agosti and F. Tomasi, Eds. CLEUP, December 2013, pp. 69–80.

[232] A. Di Iorio, M. Schirinzi, F. Vitali, and C. Marchetti, "A natural and multi-layered approach to detect changes in tree-based textual documents," in *Enterprise Information Systems*, ser. Lecture Notes

in Business Information Processing, J. Filipe and J. Cordeiro, Eds. Springer Berlin Heidelberg, 2009, vol. 24, pp. 90–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01347-8_8

[233] G. Barabucci, A. Di Iorio, S. Peroni, F. Poggi, and F. Vitali, "Annotations with EARMARK in practice: a fairy tale," in *Proceedings of the 1st International Workshop on Collaborative Annotations in Shared Environment: metadata, vocabularies and techniques in the Digital Humanities (DH-CASE), Florence, Italy.* ACM, September 2013, pp. 11:1–11:8. [Online]. Available: http://doi.acm.org/10.1145/2517978.2517990

[234] S. Peroni, F. Poggi, and F. Vitali, "Tracking changes through EARMARK: a theoretical perspective and an implementation," in *Proceedings of the 1st International Workshop on: Document Changes: Modeling, Detection, Storage and Visualization (DChanges), Florence, Italy.* ACM DocEng, September 2013.

[235] F. Meyerer and O. Hummel, "Towards plug-and-play for component-based software systems," in *Proceedings of the 19th International Doctoral Symposium on Components and Architecture (WCOP), Marcq-en-Bareul, France.* New York, NY, USA: ACM, June-July 2014, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2601328.2601334

[236] M. Piotrowski, *Natural language processing for historical texts*, ser. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012, vol. 5, no. 2.

[237] N. White, "Training tesseract for ancient Greek OCR," *Eutypon*, vol. 28–29, pp. 1–12, October 2012.

[238] B. Robertson, C. Dalitz, and F. Schmitt, "Automated page layout simplification of Patrologia graeca," in *Proceedings of the 1st International Conference on Digital Access to Textual Cultural Heritage (DATeCH '14), Madrid, Spain.* New York, NY, USA: ACM, May 2014, pp. 167–172. [Online]. Available: http://doi.acm.org/10.1145/2595188.2595213

[239] F. Boschetti, M. Romanello, A. Babeu, D. Bamman, and G. Crane, "Improving OCR accuracy for classical critical editions," in *Research and Advanced Technology for Digital Libraries*, ser. Lecture Notes in Computer

Science, M. Agosti, J. Borbinha, S. Kapidakis, C. Papatheodorou, and G. Tsakonas, Eds. Springer Berlin Heidelberg, 2009, vol. 5714, pp. 156–167. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04346-8_17

[240] A. M. Del Grosso and F. Boschetti, "Parallel OCR for ancient Greek critical editions," in *Science and Supercomputing in Europe, Research Highlights*. HPC-Europa2, 2012.

[241] J. He, Q. D. M. Do, A. C. Downton, and J. H. Kim, "A comparison of binarization methods for historical archive documents," in *Proceedings of 8th International Conference on Document Analysis and Recognition (ICDAR), Seoul, South Korea*, vol. 1. Washington, DC, USA: IEEE Computer Society, August-September 2005, pp. 538–542. [Online]. Available: http://dx.doi.org/10.1109/ICDAR.2005.3

[242] A. M. Del Grosso, S. Marchi, F. Murano, and L. Pesini, "A collaborative tool for philological research: experiments on Ferdinand de Saussure's manuscripts," in *Proceedings of revised papers of the 2nd Annual Conference of the Associazione per l'Informatica Umanistica e la Cultura Digitale. Collaborative Research Practices and Shared Infrastructures for Humanities Computing, (AIUCD) Padova*, M. Agosti and F. Tomasi, Eds. Padova: CLEUP, December 2013, pp. 163–175.

[243] A. M. Del Grosso and S. Marchi, "Una applicazione web per la filologia computazionale: un esperimento su alcuni scritti autografi di Ferdinand de Saussure," in *Guida per una edizione digitale dei manoscritti di Ferdinand de Saussure*, D. Gambarara and M. P. Marchese, Eds. Edizioni dell'Orso, 2013, pp. 131–157.

[244] A. M. Del Grosso, "Indexing techniques and variant readings management," *Studia graeco-arabica*, vol. 3, pp. 209–230, 2013.

[245] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[246] M. Manca, L. Spinazzè, P. Mastandrea, L. Tessarolo, and F. Boschetti, "Musisque deoque: Text retrieval on critical editions," in *Proceedings of workshop on Annotation of Corpora for Research in the Humanities, Heidelberg, Ger-*

many, F. Mambrini, M. Passarotti, and C. Sporleder, Eds., vol. 26, no. 2. JLCL, 2012, pp. 127–138.

[247] L. Pesini, A. M. Del Grosso, and A. Bozzi, "F. de Saussure e la linguistica romanza. Un'applicazione web per l'edizione elettronica dei manoscritti," in *Proceedings of 27th Congrès international de linguistique et de philologie romanes. Section 16: Projets en cours ; ressources et outils nouveaux,(CILPR), Nancy, France*, A. Lemaréchal, P. Koch, and P. Swiggers, Eds., Laboratoire ATILF (CNRS and Université de Lorraine) et la Société de linguistique romane. ATILF, July 2013.

[248] Y. Bizzoni, F. Boschetti, R. Del Gratta, H. Diakoff, M. Monachini, and G. Crane, "The making of ancient Greek WordNet," in *Proceedings of the 9th Language Resources and Evaluation Conference (LREC), Reykjavik, Iceland*, N. Calzolari, K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, and S. Piperidis, Eds. European Language Resources Association (ELRA), May 2014, pp. 1140–1147.

[249] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval: The Concepts and Technology behind Search*, 2nd ed. Harlow: Addison-Wesley Professional, 2011.

[250] E. Pierazzo, "Digital documentary editions and the others'," *Scholarly Editing*, vol. 35, 2014. [Online]. Available: http:www.scholarlyediting.org/2014/essays/essay.pierazzo.html

[251] F. Boschetti, "La localizzazione del Perseus Project in lingua italiana," *Quaderni DigiLab*, vol. 3, no. 1, pp. 221–234, 2014.

[252] ——, "Acquisizione e creazione di risorse plurilingui per gli studi di filologia classica in ambienti collaborativi," in *Collaborative Research Practices and Shared Infrastructures for Humanities Computing. Proceedings of revised papers of the 2nd Annual Conference of the Associazione per l'Informatica Umanistica e la Cultura Digitale (AIUCD), Padova*, M. Agosti and F. Tomasi, Eds. CLEUP, December 2013, pp. 55–68.

[253] A. Bellandi, A. Bellusci, A. Cappelli, and E. Giovannetti, "Graphic visualization in literary text interpretation," in *Proceedings of 18th Interna-*

tional Conference on Information Visualisation (IV), University of Paris Descartes, Paris, France, July 2014, pp. 392–397.

[254] D. Fabbri, “Sistema knowledge based interattivo per la gestione visuale dell’apparato critico di un testo,” Master’s thesis, Universitá degli studi di Siena, facoltá di ingegneria, Siena, 1998-1999.

[255] A. Bozzi, M. S. Corradini, and B. Tellez, “The EUMME project: towards a new philological workstation,” in *Proceedings of the 9th International Conference on Electronic Publishing (ICCC), Leuven, Belgium*, June 2005, pp. 139–144.

[256] P. Canettieri, G. Santini, M. Rovetta, and V. Loreto, “Philology and information theory : Towards an integrated approach,” in *The evolution of texts: confronting stemmatological and genetical methods*, C. Macé, P. Baret, A. Bozzi, and L. Cignoni, Eds., vol. 24–25, Proceedings of the international workshop held in Louvain-la-Neuve on September 1-2, 2004. Pisa-Roma: Istituti editoriali e poligrafici internazionali, June 2006, pp. 109–126. [Online]. Available: http://digital.casalini.it/an/2212902

[257] M. Abrate, A. M. Del Grosso, E. Giovannetti, A. Lo Duca, A. Marchetti, L. Mancini, I. Pedretti, and S. Piccini, “Il progetto Clavius on the Web: tecnologie linguistico - semantiche al servizio del patrimonio documentale e degli archivi storici,” in *Book of Abstracts del 3° convegno dell’Associazione Italiana Informatica Umanistica e Cultura Digitale. La metodologia della ricerca umanistica nell?ecosistema digitale, (AIUCD), Bologna*, F. Rossi and F. Tomasi, Eds., September 2014.

[258] A. Bellandi, A. Bellusci, E. Carniani, and E. Giovannetti, “Content elicitation: Towards a new paradigm for the analysis and interpretation of texts,” in *Proceedings of the 13th IASTED International Conference on Software Engineering, Innsbruck, Austria*, Innsbruck, February 2014.

[259] F. Mambrini, M. Passarotti, and C. Sporleder, Eds., *Annotation of Corpora for Research in the Humanities*, Proceedings of the ACRH Workshop, Journal for Language Technology and Computational Linguistics, Heidelberg, 5 January. Heidelberg: JLCL, 2011.

[260] G. Crane, "Generating and parsing classical Greek," *Literary and Linguistic Computing*, vol. 6, no. 4, pp. 243–245, 1991. [Online]. Available: http://llc.oxfordjournals.org/content/6/4/243.abstract

[261] C. Oravecz and P. Dienes, "Efficient stochastic part-of-speech tagging for hungarian." in *Proceedings of the 3th international conference on Language Resources and Evaluation (LREC), Las Palmas, Canary Islands, Spain.* European Language Resources Association (ELRA), May 2002, pp. 710–717.

[262] D. Bamman, M. Passarotti, R. Busa, and G. Crane, "The annotation guidelines of the Latin dependency Treebank and Index Thomisticus Treebank: the treatment of some specific syntactic constructions in Latin." in *Proceedings of the 6th international conference on Language Resources and Evaluation (LREC), Marrakech, Morocco*, N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, and D. Tapias, Eds. European Language Resources Association (ELRA), May 2008, pp. 71–76.