Towards a flexible open-source software library for multi-layered scholarly textual studies

An Arabic case study dealing with semi-automatic language processing

Angelo Mario Del Grosso Ouafae Nahli
Istituto di Linguistica Computazionale "A. Zampolli" (ILC)
Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy
angelo.delgrosso@ilc.cnr.it ouafae.nahli@ilc.cnr.it

***Abstract*—This paper presents both the general model and a case study of the Computational and Collaborative Philology Library (CoPhiLib), an ongoing initiative underway at the Institute for Computational Linguistics (ILC) of the National Research Council (CNR), Pisa, Italy. The library, designed and organized as a reusable, abstract and open-source software component, aims at solving the needs of multi-lingual and cross-lingual analysis by exposing common Application Programming Interfaces (APIs). The core modules, coded by the Java programming language, constitute the groundwork of a Web platform designed to deal with textual scholarly needs. The Web application, implemented according to the Java Enterprise specifications, focuses on multi-layered analysis for the study of literary documents and related multimedia sources. This ambitious challenge seeks to obtain the management of textual resources, on the one hand by abstracting from current language, on the other hand by decoupling from the specific requirements of single projects. This goal is achieved thanks to methodologies declared by the "agile process", and by putting into effect suitable use case modeling, design patterns, and component-based architectures. The reusability and flexibility of the system have been tested on an Arabic case study: the system allows users to choose the morphological engine (such as AraMorph or Al-Khalil), along with linguistic granularity (i.e. with or without declension). Finally, the application enables the construction of annotated resources for further statistical engines (training set).**

Keywords—API Design; Information Engineering; Design Patterns; Text Processing; Arabic Natural Language Processing

I. Introduction

The Computational Philology Laboratory (CoPhi Lab) at the Institute for Computational Linguistics (CNR), Pisa, Italy is currently working on several collaborative and computational projects for literary studies and for classical and modern language analysis [1-3][10][12].

On the basis of our experience and of authoritative scientific papers [4][6-9][11][13-14][16-18], this field of the digital humanities has shown to require programming interfaces and abstract data structures for multi-lingual and multi-witness texts [16]. Since literary cross-lingual studies are constantly growing and evolving, both in number and in complexity, related soundness applications have become more and more demanding [6][8][17].

Therefore, it is important to develop shared systems able to analyze, compare and link such resources [2][18]. Studies performed by an abstract framework allow to reuse early results, to facilitate interoperability and integration, and to investigate worthy parallel phenomena [2][9][14][17-18].

Current software systems for scholarly studies generally depend on single initiatives and language requirements; thus, each language requires a different system for language analysis. Instead, computational projects for worthy cultural heritage cannot be addressed by specific needs or languages, even less can they rely on rigid and monolithic applications [2-3][8-9][13-14]. A modular and abstract framework would allow developers, on the one hand, to reuse the general model and the core components and, on the other hand, to add functionalities and to solve new requirements by plugging extensions into the system [3].

Therefore, the Application Programming Interfaces (APIs) provide a means of access to ad-hoc implementations for required services (such as linguistic morphological analysis in Arabic by Buckwalter's engine).

The aim of this paper is both to introduce the methods adopted to achieve those scientific purposes and to illustrate the Arabic case study chosen to test the validity of the architecture.

This work is the result of a trade-off between a bottom-up approach and a top-down design [3]. The initiative is bottom-up because it refines the framework model and the abstract interfaces thanks to already implemented functionalities for previous works. The top-down design provides developers with the abstract entities and the core APIs. The ultimate aim is to realize a flexible and reusable system able to detect the resource language, to choose the morphological engine, to perform analyses at different granularities, to manage the repository, and to construct training sets.

The importance of this project is confirmed by the many initiatives and tools in the fields of literary and linguistic computing and of natural language processing for cultural heritage. These include a number of reference projects and supranational infrastructures such as Clarin, DARIAH, and TextGrid, as well as systems within academic works. Alpheios and Perseid, adopted by the Perseus project, can be considered two of the most significant systems. The European COST Action Interedition has obtained prominent results like those of CollateX; on the other hand, the Moruca framework and the Pundit Semantic tool have received great attention in the last few years. Initiatives highlighting the work of TEI, such as Van Gogh and Tustep, are at the state-of-the-art. Finally, with regard to effective software libraries, Java open-source efforts such as Lucene, Tika, UIMA, LingPipe, Gate, StandfordNLP, and OpenNLP are worth mentioning. This overview cannot obviously cite all the ecosystems available. However, the long-standing challenge for almost all of these works lies in multi-lingual handling, actual modularity, reusability and scalability.

II. Methodology

A. Requirements and Specifications

CoPhiLib is formed by modular and extendible components organized in different architectural layers. The component details have been modelled by adopting UML diagrams to describe functional specifications and user requirements. The core component provides general programming interfaces (behavior), and also manages abstract data structures and domain entities (ADTs). Flexibility is reached by applying general and proven software engineering principles and object-oriented solutions to face recurring problems (i.e. separation of concerns, abstraction, polymorphism, information hiding, design patterns). Furthermore, new components, plug-ins and extensions add extra functionalities and specific requirements by means of registration and/or negotiation techniques. Finally, use-case driven process, mockup storyboard, and agile development guidelines allow progressive enhancements and iterative refinements.

The Web Applications developed exploit the Cophi framework and provide suitable Graphical User Interfaces (GUIs) that allow users to edit, visualize and process resources. Moreover, the system controllers can rely on abstract methods for processing resources, thanks to the availability of Component Interfaces (APIs).

The first phase of the design process is the analysis of requirements, which identifies functionalities, actors and system components. In such a process, several UML diagrams tie down the specifications for the application. Basically, the use case diagram defines user interactions; the component diagram traces the core blocks of the system; and the class diagrams reveal critical domain entities and constitutive relations.

The use case diagram in Fig. 1 illustrates four types of actors and the ways in which they interact with the system. Two actors exploit the capabilities of the components either generally or specifically, while the others can configure and extend the system. The general user visualizes and interacts with data, and also performs basic and advanced searches. The domain expert user specializes the general user (i.e. he has the basic functionalities). Furthermore, he can manage the primary sources (resource digitization/acquisition, text and related multimedia), process resources (e.g. linguistic analysis, proofreading, etc.), establish both internal and external relations among entities, perform data CRUD operations (i.e. Create, Read, Update, and Delete annotations and comments related to the content). Finally, the user handles the information and meta-data regarding the resources supports such as stones, rolls, papyri, etc.

The developer actor specializes the general user as well. He configures the system and adds extensions. Finally, the domain expert developer specializes both the domain expert and the developer user. He is in charge of developing extensions to the system.

Requirements that involve domain needs have been logically divided into flexible and autonomous components (Fig. 2) interacting by adequate orchestration mechanisms and Service Provider Interfaces (SPIs). The main components are illustrated by UML diagrams and are as follows:

- The *Content component* allows to represent and handle the primary sources. The resources can be represented by text and/or images.
- The *Layer component* is dedicated to the processing and analysis of resources at various levels of granularity.
- The *Editing component* modifies and updates and manages the different versions of the data.
- The *Search component* is dedicated to efficient and effective advanced searches and indexing features.
- The *Relationship component* manages connections among internal and external resources, annotations and data.
- The *Support component* deals with computational models for objects, such as stones, papyri, scrolls, which bear the textual information.
- The *Presentation component* controls all the visualization aspects and information rendering, and provides suitable user interaction tools.

The architecture of the system provides functionalities responding to domain specifications and to user requirements through components that can be used independently of the single needs. The management of textual resources, for example, can take advantage of linguistic analysis capabilities regardless of current language. Finally, it is sufficient to set up software modules which effectively implement or extend basic services of abstract functions (Information Hiding).

B. Modeling

The overall architecture (Web Technologies combined with CophiLib) is based on the well known Model-View-Controller (MVC) pattern, which ensures separation and decoupling among: (a) the representation of internal data status, (b) rendering, (c) system interaction, (d) user scenarios, and (e) content management. Specifically, CoPhiLib has been designed by adopting the object-oriented paradigm and has been developed by exploiting the Java programming language. It is formed by several packages reflecting the components described above. Each package is composed of nested sub-packages and classes derived from the object-oriented design process (OOP). Abstract and concrete classes as well as interaction among them follow component-based approach and design pattern endorsement. Each pattern satisfies peculiar requirements, which are as follows: Factory, Strategy, Command, Data Transfer Object (DTO), Data Access Object (DAO), Delegate, Adapter. Static and dynamic UML diagrams describe the model entities, their behavior and relations. This paper highlights part of the functionalities of the Layer component (Fig. 2). In general those functionalities are concerned with resource processing, data mining, content analysis and annotation. Flexibility is achieved through factory and delegation patterns towards sub-components that perform the correct task by choosing the right strategy. The sequence of calls activates Layer APIs which invoke concrete routines exposed by Layer sub-components.

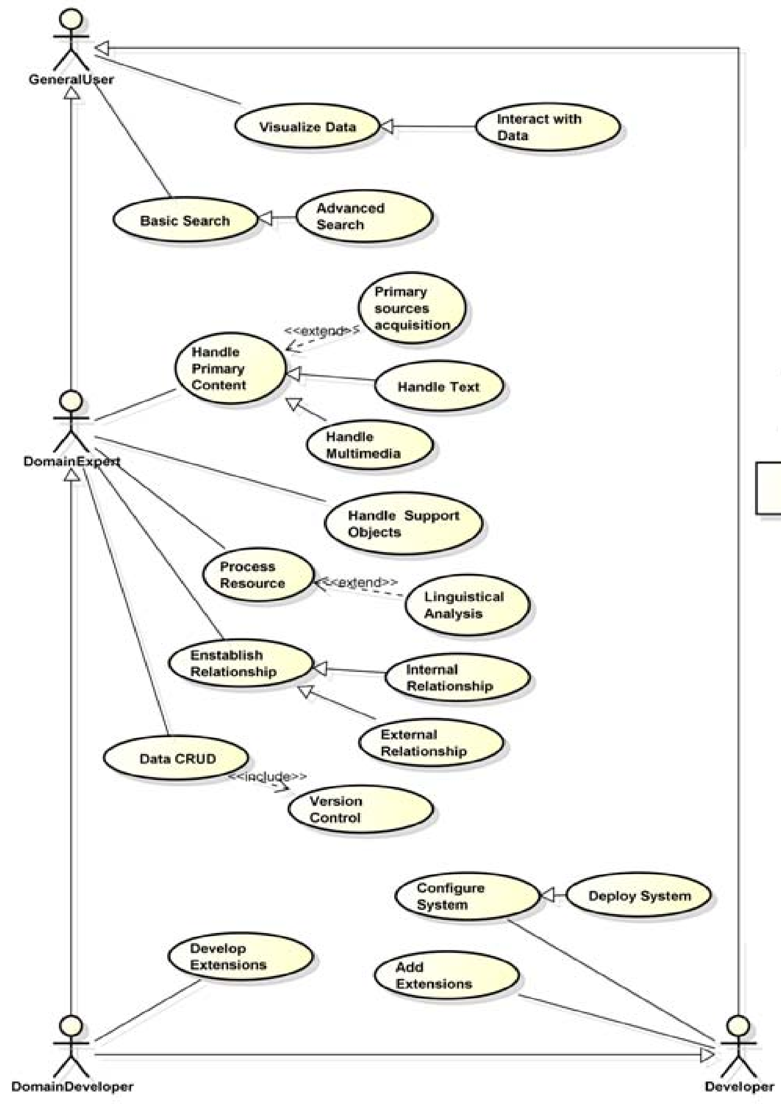


Fig. 1. Basic CophiLib use-cases

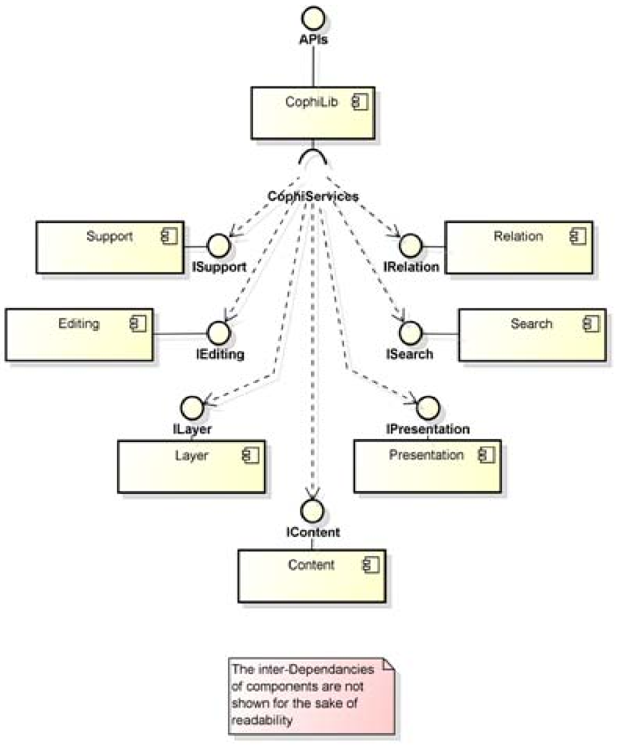


Fig. 2. Basic CophiLib components

Fig. 3 illustrates the objects (class diagram) and the behavior (sequence diagram) that implement text analysis. Therefore, the library has to provide mechanisms to substitute processing engines and adapt analysis strategies. The application remains consistent, thanks to indirection techniques and adapter patterns, when legacy or third party tools are exploited. Cophi APIs expose functionalities for linguistic analysis by means of language independent entities that dynamically instantiate the appropriate tools. The sequence diagram shows the terms of the mechanism. The analysis objects provide methods that require the content and return its analysis. The external entities can interact only with the *CophiLinguisticAnalysis* public component, which creates and uses a special object called *AnalysisContext*. *AnalysisContext* makes the right association among language, linguistic analysis and the engine to be used. For the sake of modularity, *AnalysisContext* uses the *EngineCreator* object which returns the reference to the engine, and it invokes the method for content analysis. The working principles expressed in Fig. 3 are further clarified in the following Arabic case study.

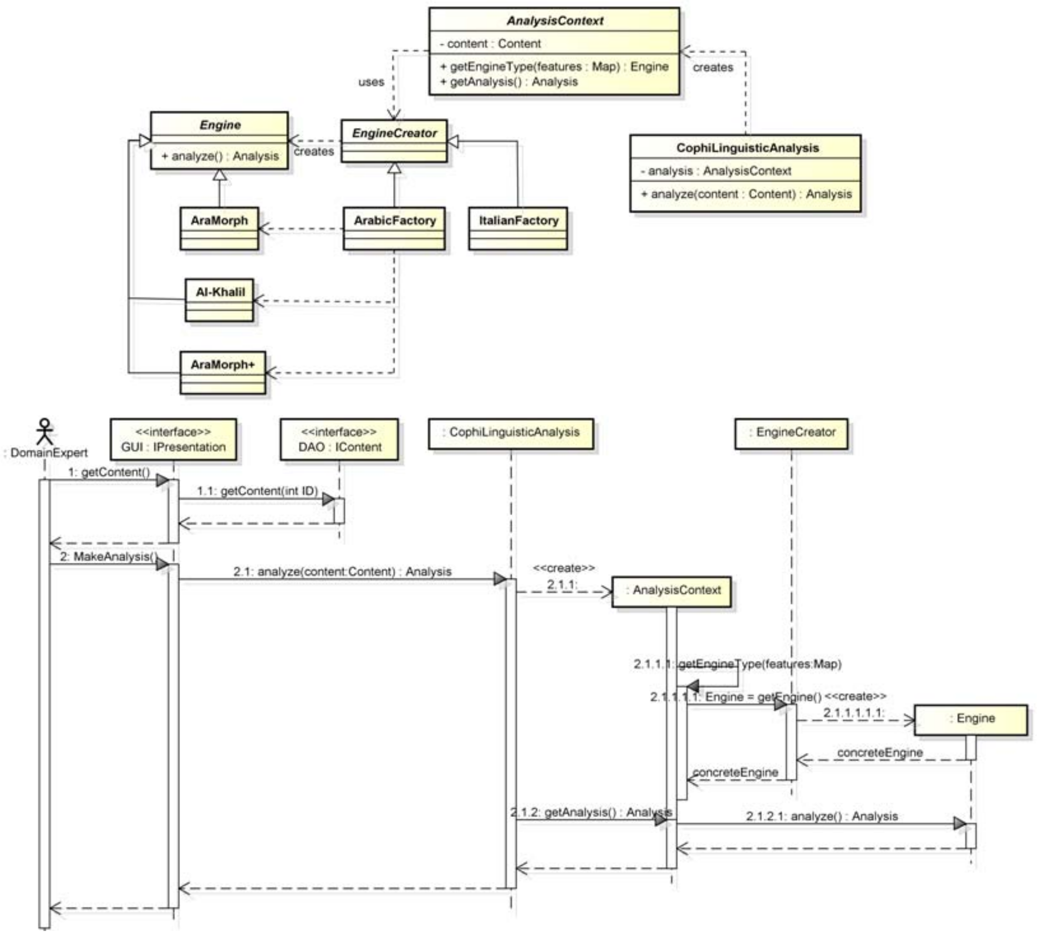
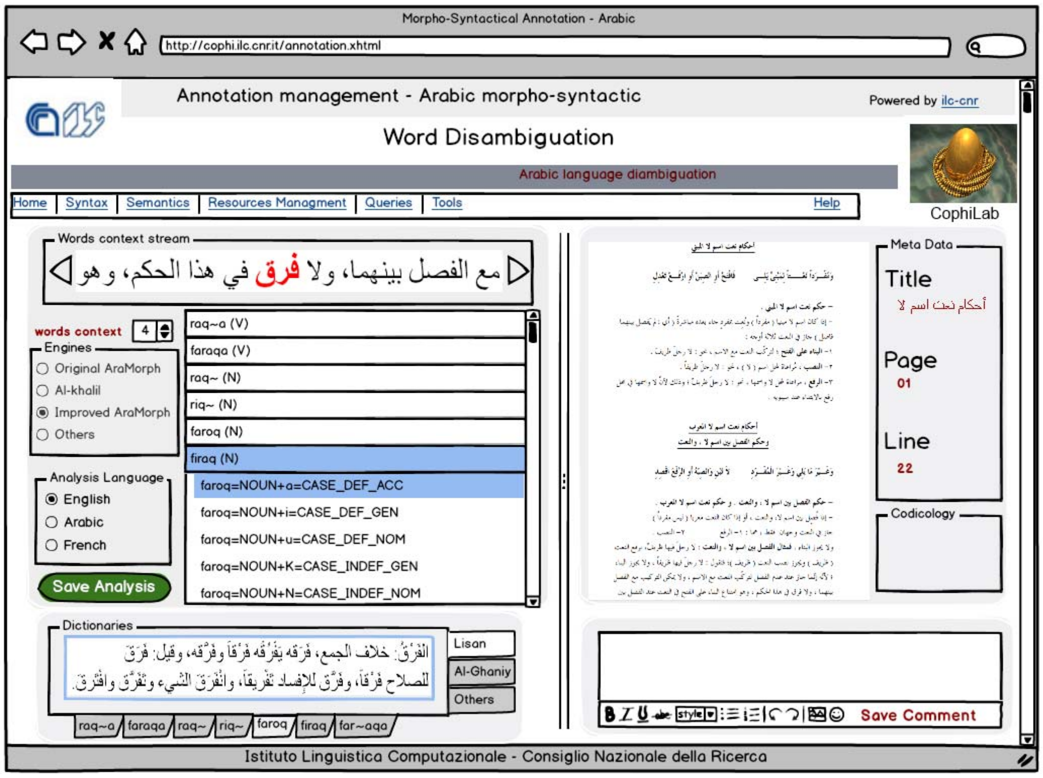

AnalysisContext
- content : Content
+ getEngineType(features : Map) : Engine
+ getAnalysis() : Analysis
uses
creates
Engine
+ analyze() : Analysis
creates
EngineCreator
CophiLinguisticAnalysis
- analysis : AnalysisContext
+ analyze(content : Content) : Analysis
AraMorph
ArabicFactory
ItalianFactory
Al-Khalil
AraMorph+
: DomainExpert
<<interface>> GUI : IPresentation
<<interface>> DAO : IContent
: CophiLinguisticAnalysis
: EngineCreator
1: getContent()
1.1: getContent(int ID)
2: MakeAnalysis()
2.1: analyze(content:Content) : Analysis
<<create>> 2.1.1:
: AnalysisContext
2.1.1.1: getEngineType(features:Map)
2.1.1.1.1: Engine = getEngine() <<create>>
2.1.1.1.1.1:
: Engine
concreteEngine
concreteEngine
2.1.2: getAnalysis() : Analysis
2.1.2.1: analyze() : Analysis

Fig. 3. Class and Sequence diagrams for cross-lingual analysis


Morpho-Syntactical Annotation - Arabic
http://cophi.ilc.cnr.it/annotation.xhtml
Annotation management - Arabic morpho-syntactic
Powered by ilc-cnr
Word Disambiguation
Arabic language diambiguation
Home | Syntax | Semantics | Resources Managment | Queries | Tools | Help
CophiLab
Save Analysis
Save Comment
Istituto Linguistica Computazionale - Consiglio Nazionale della Ricerca

Words context stream

مع الفصل بينهما، ولا فرق في هذا الحكم، وهو

words context 4

Engines
- ○ Original AraMorph
- ○ Al-khalil
- ◉ Improved AraMorph
- ○ Others

Analysis Language
- ◉ English
- ○ Arabic
- ○ French

raq~a (V)
faraqa (V)
raq~ (N)
riq~ (N)
faroq (N)
firaq (N)
faroq=NOUN+a=CASE_DEF_ACC
faroq=NOUN+i=CASE_DEF_GEN
faroq=NOUN+u=CASE_DEF_NOM
faroq=NOUN+K=CASE_INDEF_GEN
faroq=NOUN+N=CASE_INDEF_NOM

Meta Data
Title
أحكام نعت اسم لا
Page
01
Line
22
Codicology

Dictionaries

الفَرْقُ: خلاف الجمع، فَرَقه يَفْرُقه فَرْقاً وفَرَّقه، وقيل: فَرَقَ للصلاح فَرْقاً، وفَرَّق للإفساد تَفْريقاً، وانْفَرَقَ الشيء وتَفَرَّقَ وافْتَرَقَ.

Lisan
Al-Ghaniy
Others

raq~a | faraqa | raq~ | riq~ | faroq | firaq | far~aqa

Fig. 4. Graphical User Interface under development for the CophiLib Web Application (Arabic use case)

III. RESULTS: THE ARABIC CASE STUDY

Fig. 4 represents a customized mock-up Web GUI for Arabic needs. It explains the user scenario related to the UML diagrams of Fig. 3.

A domain expert user asks the controller interface for content. The controller, in turn, calls the data management function to retrieve the selected resource. The content (text and meta-data) is visualized in a structured way. Another user's action then activates the linguistic analysis procedure. At this point, the creation of the *AnalysisContext* object allows the system to detect the Arabic language and the available linguistic engines that can be used. The domain expert user selects the analysis configuration desired (actual engine, labelling features, tag-set, granularity).

The morphological engine used in this work is Aramorph[1], which uses three separate "dictionaries" for prefixes, suffixes and stems respectively, and three tables that control their morphological compatibility. First of all, the input word is segmented into prefix, stem, and suffix, which are checked through embedded dictionaries. The next step is to determine whether or not they are compatible, on the basis of their morphological characteristics. If the compatibility condition is verified, the analysis is accepted. Aramorph makes it possible for the tokenization process, morphological analysis and lemmatization to take place.

The engine does not handle the nominal case and verbal mood. To solve these matters, several updates have been performed to solve the shortcomings of Aramorph [12], and to improve the performance of the engine.

The system allows users to choose the morphological original engine (Aramorph) or the improved one. Furthermore, another engine called Alkhalil[2] [5] has been made available as possible option. With regard to analysis granularity, the system make it possible to perform lemmatization while, optionally, the analysis is enriched by the morpho-syntactic tagging.

In Arabic, the vowel in final position defines the nominal cases (except for the dual and "sound" plural). The case markers are three:

- Nominative: The nominative suffix is the vowel ضَمَّة *ḍamma* /u/ for the definite words or تَنْوِين ضَمَّة *tanwīn ḍamma* "*un*" for the indefinite words.
- Accusative: The accusative suffix is the vowel فَتْحَة *fatḥa* /a/ for the definite words or تَنْوِين فَتْحَة *tanwīn fatḥa* "*an*" for the indefinite words.
- Genitive: : The genitive suffix is the vowel *kasra* /i/ for the definite words or تَنْوِين كَسْرَة *tanwīn kasra* "*an*" for the indefinite words.

Therefore, there are six possible ways in which the grammatical cases of an Arabic noun can be represented, for example Table 1 reports the noun فَرْق ***farq*** "difference".

TABLE I. DECLENSION OF THE NOUN فَرْق "DIFFERENCE"

Declension of the noun فَرْق ***farq***	***Nominative case***	***Accusative case***	***Genitive case***
definite	فَرْقُ farq-u	فَرْقَ farq-a	فَرْقِ farq-i
indefinite	فَرْقٌ farq-un	فَرْقًا farq-an	فَرْقٍ farq-in

Also the short vowel of the imperfective verb defines the verbal mood (except for the second and third persons dual and plural). For example[3]:

1- يَكْتُبُ

ya-ktub-u

IPFV.3-write-PRS.IND.3SGM

'he writes'

2- لَنْ يَكْتُبَ

lan ya-ktub-a

NEG IPFV.3-write-FUT.IND

'He will not write'

3- أَنْ يَكْتُبَ

'an ya-ktub-a

That IPFV.3- write -PRS.SUB

'that he writes'.

4- لَم يَكْتُبْ

lam ya-kotub-Ø

NEG IPFV.3-write--PST.IND

'he did not write'.

5- فَلْيَكْتُبْ

fa=l=ya-kotub-Ø

therefore=that=IPFV.3SM-write-JUSSIVE=it.3SM.ACC

'therefore that he write it".

The absence of vowels makes morpho-syntactic analysis more difficult, since the results proposed by any morphological engine are more ambiguous [15].

For example, the word ***frq*** has analyses potentially referred to four nominal lemmas (***faroq***, ***firaq***, ***riq~***, and ***raq~***) and four verbal lemmas (***raq~a***, ***faraqa***, ***fariqa*** and ***far~aqa***). The total number of possible analyses is twenty-six. Table 2 shows an example of analysis regarding a nominal lemma, while Table 3 illustrates the analysis of a verbal lemma.

[1] http://nongnu.org/aramorph/english/index.html

[2] http://alkhalil.sourceforge.net

[3] In this paper, interlinear glosses follow the standard set of parsing conventions and grammatical abbreviations explained in the Leipzig Glossing Rules: "The Leipzig Glossing Rules: Conventions for interlinear morpheme-by-morpheme glosses" February 2008. Segmentable morphemes are separated by hyphens and clitics by an "equals sign", in both transliterated language and interlinear gloss.

TABLE II. LEMMA *faroq* ANALYSES

lemme	***morpho-syntactic analysis***
faroq	faroq=NOUN+a=CASE_DEF_ACC
	faroq=NOUN+i=CASE_DEF_GEN
	faroq=NOUN+K=CASE_INDEF_GEN
	faroq=NOUN+N=CASE_INDEF_NOM
	faroq=NOUN+u=CASE_DEF_NOM

TABLE III. LEMMA *far~aqa* ANALYSES

far~aqa	far~aq=VERB_PERFECT+a=PVSUFF_SUBJ:3MS+
	fur~iq=VERB_PERFECT+a=PVSUFF_SUBJ:3MS+

In order to face and solve this problem, the system allows the user to split the work into two phases independently of the linguistic engine adopted. Firstly, it is possible to establish the lemma corresponding to the presented word, according to the textual context. This phase of lemmatization is aided by visualizing dictionary entries (e.g. Lisan, Al-Ghany) that are embedded into the system (Fig. 4). Secondly, the user highlights the declension or flexion related to the selected lemma. The domain expert user determines whether it is sufficient to have only the lemmatization or to complete the entire process by adding the morpho-syntactic features. Such a flexibility is achieved according to the specifications of the aforementioned abstract API and the appropriate design patterns shown in Fig. 3 (MVC, Factory, Delegation, Strategy, DTO, DAO). Disambiguation of the linguistic analysis has been carried out by means of filters based on grammatical rules. Textual resources, which are reviewed and manually annotated, progressively enrich the training set available in order to improve contextual stochastic processing.

IV. CONCLUSION

In this work we have illustrated the initiative that we are carrying out at the Cophi Lab (ILC, CNR, Pisa), and which is aimed at the design and release of a modular library along with its relative Application Programming Interfaces (APIs). The goal of this ambitious project is to solve multi-lingual and cross-lingual problems of scholars using a decoupling, modularity, flexible and re-usable system, with particular regard to the "separation of concerns" approach. Implementation is made possible by exploiting the abstract components developed in Java language and designed according to well-known patterns and UML diagrams. The overall architecture is based on the library of components, which is the core of a Web platform dealing with texts, related sources and linguistic analysis. The Arabic case study shows the benefit of the proposal, addressing the actual user scenario in a linguistic analysis work. Finally, the modularity of the system makes it possible to reuse the components in many applications. This aim is achieved by pursuing the "Open-Close" principle: Components should be open for extension, but closed for modification.

ACKNOWLEDGMENT

The authors wish to thank Laura Cignoni (ILC, CNR, Pisa) for the English review of the manuscript.

REFERENCES

[1] M. Abrate et al., "Sharing cultural heritage: the Clavius on the Web project," in Proceedings of the 9th LREC, 2014, pp. 627-634.

[2] A. Bozzi, "G2A: a Web application to study, annotate and scholarly edit ancient texts and their aligned translations," in Stuida graeco-arabica vol. 3, ERC Ideas 249431, Pisa: Pacini, 2013, pp. 159-171.

[3] F. Boschetti, A.M. Del Grosso, A.F. Khan, M. Lamé, O. Nahli, "A top-down approach to the design of components for the philological domain," in book of abstracts of DH'14, 2014.

[4] F. Boschetti, M. Romanello, A. Babeu, D. Bamman, G. Crane, "Improving OCR accuracy for classical critical editions," in Proceedings of the 13th ECDL'09, M. Agosti, J. Borbinha, S. Kapidakis, C. Papatheodorou, G. Tsakonas eds., Berlin: Springer-Verlag, 2009, pp. 156-167.

[5] A. Boudlal et al., "Alkhalil Morpho Sys: A morphosyntactic analysis system for Arabic texts," in Proceedings of the 11th International Arab Conference on Information Technology (ACIT'10), 2010.

[6] L. Burnard, "The evolution of the Text Encoding Initiative: from research project to research infrastructure," in Journal of the Text Encoding Initiative, Issue 5, 2013, DOI: 10.4000/jtei.811.

[7] D. Buzzetti, "Digital editions and text processing," in text editing, print and the digital world, M. Deegan, K. Sutherland. eds., Surrey: Ashgate, 2009, pp. 45-61.

[8] G. Crane at al., "Student researchers,citizen scholars and the trillion word library," in Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'12), New York: ACM, 2012, pp. 213-222.

[9] G. Crane, A. Lüdeling, "Introduction to the special issue on corpus and computational linguistics, philology, and the linguistic heritage of humanity,", in Journal on computing and cultural heritage, Vol 5 (1), New York: ACM, 2012, pp. 1-5.

[10] A. Del Grosso, S. Marchi, "Una applicazione web per la filologia computazionale. Un esperimento su alcuni scritti autografi di Ferdinand de Saussure," in Guida per un'edizione digitale dei manoscritti di Ferdinand de Saussure, D. Gambarara, M. P. Marchese, eds., Alessandria: Dell'Orso Editore, 2013, pp. 131-157.

[11] D. Fiormonte, "Towards a cultural critique of the digital humanities," in Historical Social Research vol 37(3), M. Thaller ed., Köln: HSR, 2012, pp. 59-76

[12] O. Nahli, "Computational contributions for Arabic language processing Part 1. The automatic morphological analysis of Arabic texts," in Studia graeco-arabica vol. 3, ERC Ideas 249431, ed., Pisa: Pacini, 2013, pp. 195-206.

[13] P. Robinson, "The history of scholarly digital editions, Plc.," in Papers of the bibliographical society of Canada vol 51 (1), G. Little ed., Toronto: ISSN 0067-6896, 2013 pp. 83-104.

[14] P. Robinson, "Towards a theory of digital editions," in the Journal of the European Society for Textual Scholarship vol 10 W.Van Mierlo, A. Fachard eds., Amsterdam: Rodopi, 2013, pp.105-132.

[15] M. Shokrollahi-Far, "Mobin: a knowledge-based morpho-syntactic parser for Arabic," in Artificial Intelligence and Signal Processing (AISP), 16th CSI International Symposium on, Piscataway, N.J.: IEEE, 2012, pp.374-379.

[16] D. Schmidt, R. Colomb, "A data structure for representing multi-version texts online," in International Journal of Human-Computer Studies vol 67(6), Elsevier, 2009, pp. 497-514.

[17] P. Gooding, M. Terras, C. Warwick, "The myth of the new: Mass digitization, distant reading, and the future of the book," in LLC, vol 28(4), Oxford University Press, 2013, pp.629-639.

[18] K. Sutherland, E. Pierazzo, "The author's hand: from page to screen," in Collaborative research in the digital humanities, Surrey: Ashgate, 2012, pp. 191-212.