Bilkent University

Department of Computer Engineering

Senior Design Project

Etymøn: A Deep-Learning Application for Etymological Clustering of Words

Low-Level Design Report

Nashiha Ahmed, Mert İnan, Cholpon Mambetova, Utku Uçkun

Supervisor: Prof. Mehmet Koyutürk
Jury Members: Prof. Uğur Doğrusöz and Prof. Çiğdem Gündüz Demir

Low-Level Design Report
February 12, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

Low Level Design Report

Etymøn: A Deep-Learning Application for Etymological Clustering of Words

1. Introduction

Etymøn is an analysis and tracing tool for word origins in all languages. It will be used to review current etymological language families and if possible find new connections that were not already present in current taxonomy. It will accomplish this using a deep learning approach.

In the following sections, a brief description of the system and the system requirements are discussed. In addition, Etymøn's low-level design is also detailed.

1.1. Purpose of the System

Current etymological analyses rely on pattern matching or tracings between different languages by experts in linguistics [1], yet it may be cumbersome or even improbable to detect word origins in situations where direct links cannot be observed between two different words. In this case, Etymøn will pose an advantage as it will be using a large corpus of data in order to match words in any given language.

Various studies carried out by linguistic experts and historians improved the understanding of language and its origins [2]. However, there is still "room for improvement" in the field. Currently, most of the studies target the Proto-Indo-European language family [2]. There is sparse research done for other languages, and there is not a single, unified resource for this information. Most of the information is scattered online or among other forms of literature.

Since there is no similar project in the market yet, our software will be designed from scratch, which would make it a greenfield project. However, we will use other existing algorithms to build our software, such as deep learning algorithms among others that will be specified further in the report.

1.2. Object Design Trade-Offs

Our system relies on working Augmented Reality software, Object Recognition software, and word databases online. We will be reusing existing packages and libraries, since our system is complicated and building these will take more time than is allotted to us to meet the project deliverable deadline. The trade-off is that it may not be completely compatible with our system, which may lead to data inaccuracies or incompleteness.

1.3. Interface Documentation Guidelines

Naming conventions will be used to make the development and design phases cohesive and understandable by all stakeholders in the projects. The following table will detail the naming and usage conventions that will be used.

Identifier Type	Rules for Naming	Examples
Packages	Package names are always written starting with a capital letter. If package names are a combination of multiple words, each word starts with a capital letter. Abbreviation letters are all capitalized. Package names are not lengthy but concise and meaningful.	package EtymonUI;
Classes	Class names follow the same naming conventions as package names. Abbreviations are only used when universally understood but are generally avoided to convey meaning clearly. Class names can be a combination of numerals and letters but not special characters.	class WordCloud;
Interfaces	Interface names follow the same naming conventions and class names.	interface LanguageSea;
Methods	Method names are verbs unlike package, class, and interface names. Method names start with a lowercase letter with first letter of internal words capitalized. Method names also need to be meaningful and concise and appropriate to understanding the method's purpose.	createCloud();

Variables	Variable names are also generally nouns. They start with lowercase letters, as methods and internal words start with an uppercase letter. Variable names may contain special characters but must not start with special characters. Variable names too must be meaningful but concise. Variables are all be initialized to avoid potential errors.	int wordCount;
		ArrayList<String> wordList;

Constants	Constants names have letters all capitalized and initialized. Internal words are separated by underscore character.	static final int MAX_WORD=1000 00;

Whitespace	In general, whitespace is used when needed to improve clarity of code. For example, the next line is used after the introduction of a branching statement. Spaces are used after an operator but can also be put before an operator.	for(int i = 0; i < 100; i++)
		{
		createWordCloud();
		}

Indentation	Indentation in our code is 4 spaces. If code is contained within a larger element, it is indented for code clarity.	
Brackets	Brackets are put on separate lines to help visual clarity on reading and writing code segments.	

1.4. Engineering Standards

The engineering standards we will be using are as follows:

- To model our software, Unified Modeling Language (UML) is used.
- IEEE engineering standards are used throughout the implementation and design of the project.
- The ACM Code of Ethics and Professional Conduct will also be enforced. Suggestions on the enforcement plan were given in the Project Specifications Report.

1.5. Definitions, Acronyms, and Abbreviations

Some definitions of Etymøn jargon are provided.

- The *Language Sea* is the first view that the user is greeted with. It is a zoomed out map of the most abundant words graphed together to make a sea like shape.

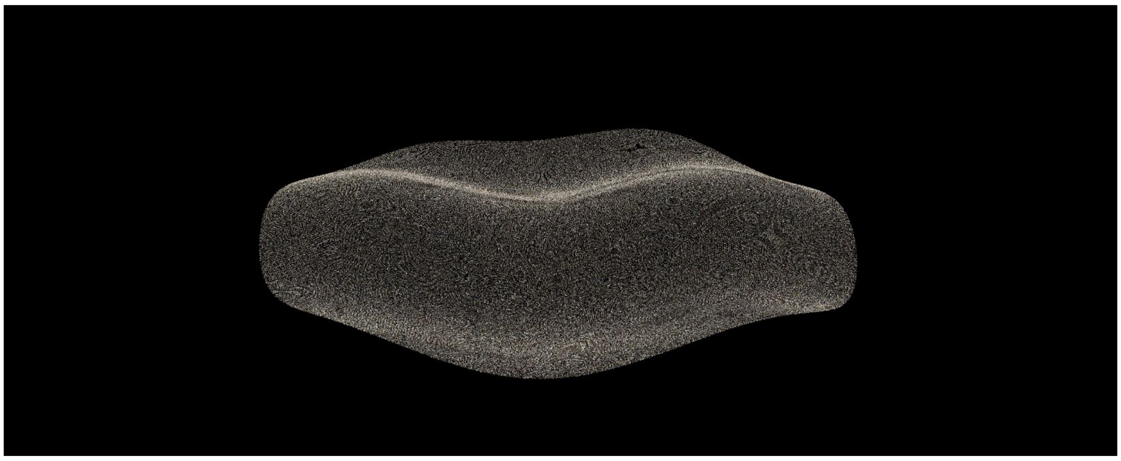

Figure 1 This figure depicts a wave-like pattern that will be like the Language Sea. [3]

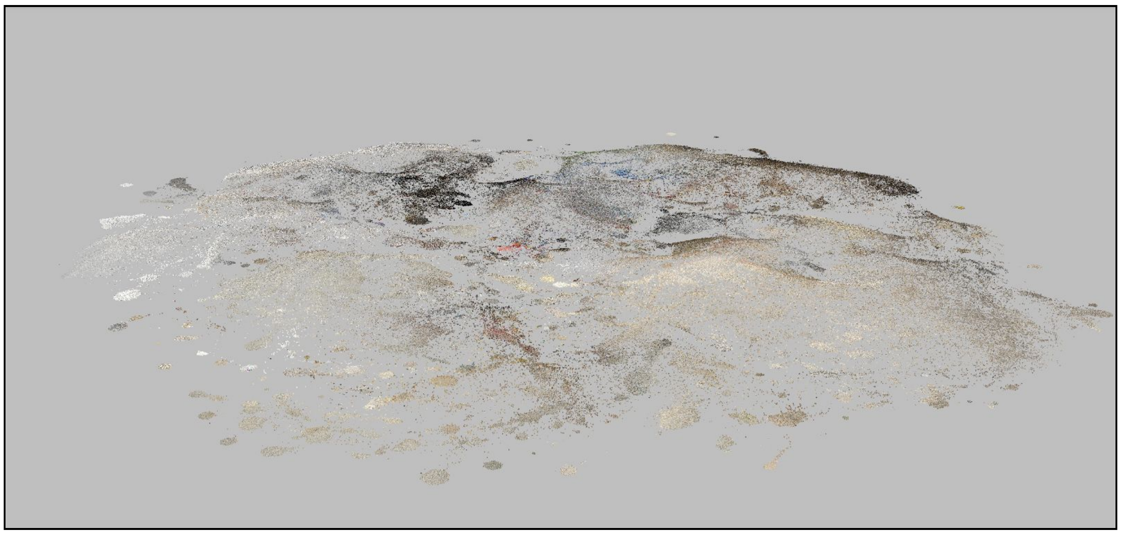

Figure 2 This figure is another clustered space that will be like the Language Sea. [3]

- The *Word Cloud* is a local graph for words clustered close to one another.

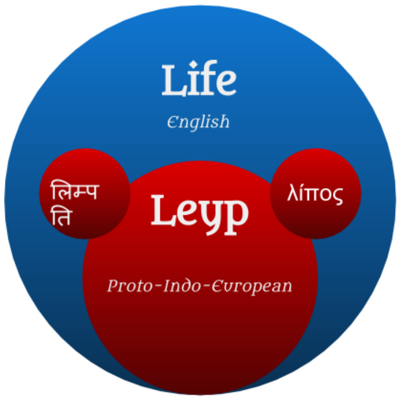

Figure 3 This figure shows a local graph for the English word, "life". Its origin is identified to be "leyp" in the Proto-Indo-European language family, and two descendent words —one in Sanskrit and one in Greek— are given next to the origin word.

2. Packages

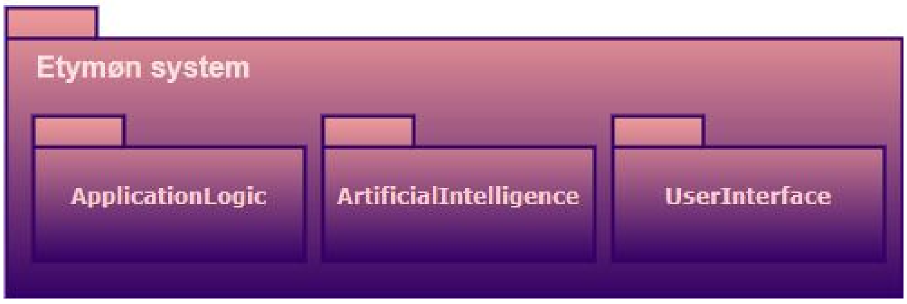

Figure 4 This figure is the UML diagram for the packages of the Etymon system.

User Interface package contains the WebGL and other graphics component codes. It includes specifically the mobile interface, web interface. Artificial Intelligence package represents the artificial intelligence portion of the whole system. It includes the Word Database and the Machine Learning related classes. Application Logic package represents the application logic of the Etymon system. It includes Augmented Reality Management, Query Management, and Object Recognition.

3. Class Interfaces

In this section of the report, we will be presenting the in-depth look to individual classes of the Etymon system. Their functions and attributes are described. Several design patterns are also described in the following paragraph. Final class diagram can be seen in Figure 5 and 6.

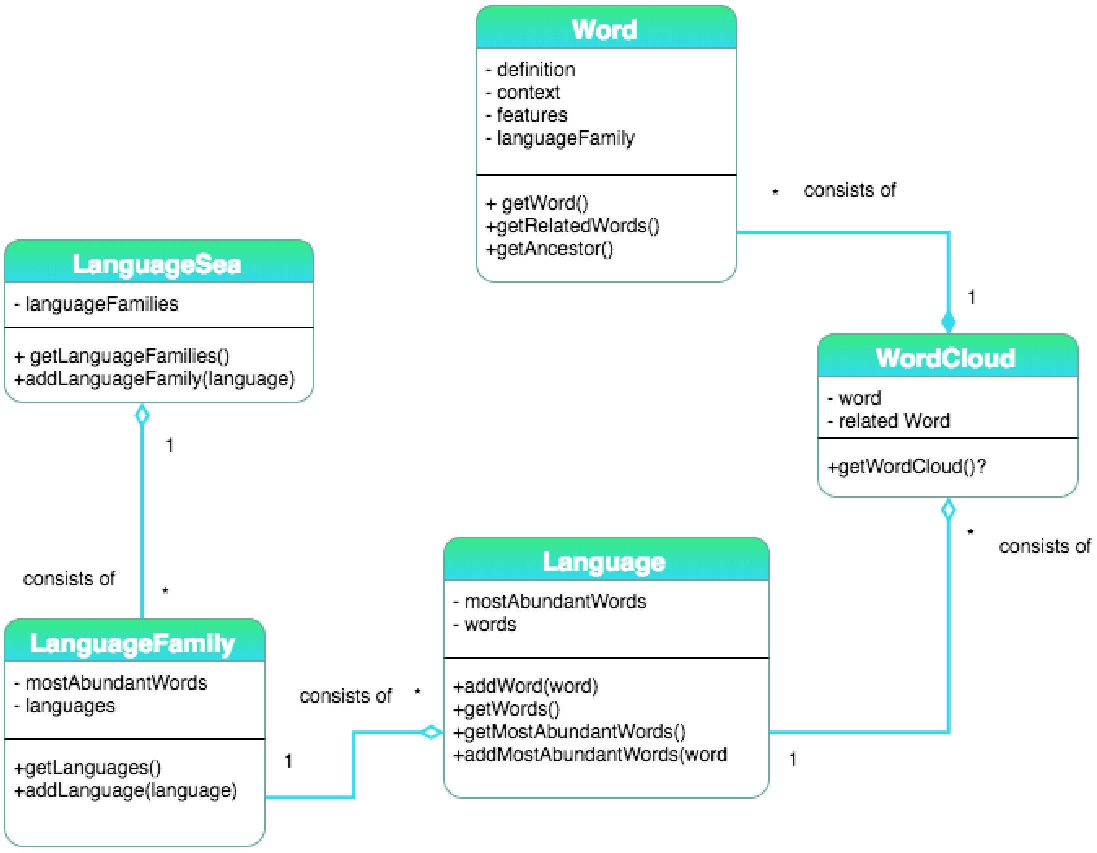

Figure 5 This figure shows the UML diagram for the entity classes of the Etymon system.

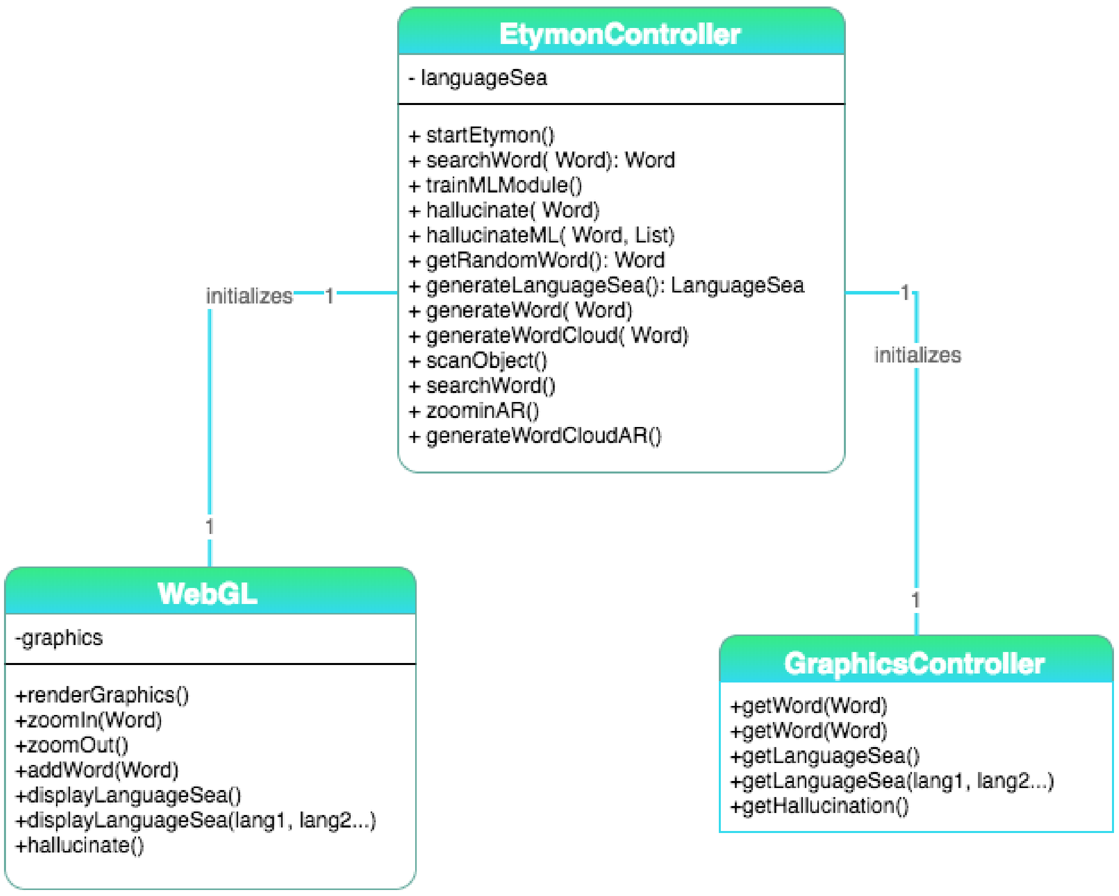

Figure 6 This is the UML class diagram representing the controller classes of Etymon.

Certain design patterns are chosen for objects of the Etymon. EtymonController and WebGL will be singleton classes as they will be initialized only once. LanguageSea object will follow an object pool pattern. Word class will be a prototype class and the GraphicsController will be an adapter class.

Class name	LanguageFamily
Attributes	mostAbundantWords: The most used words in this language family languages: Languages that are part of the specific language family.
Functions	getLanguages(): Returns the languages within the specific language family. addLanguage(newLanguage): Adds a new language to the specific language family. getMostAbundantWords(): Returns the most abundant words within the specific language family. addMostAbundantWords(word): Adds a new word to the most abundant words of specific language family.

Class name	Language
Attributes	words: all the words present in this language mostAbundantWords: The most used words in this language
Functions	addWord(word): Adds a new word to the specific language. getWords(): Returns the words within the specific language. getMostAbundantWords(): Returns the most abundant words within the specific language. addMostAbundantWords(word): Adds a new word to the most abundant words of specific language.

Class name	WordCloud
Attributes	word: The specific word that the word cloud is built upon. relatedWords: Related words to the specific word
Functions	getWordCloud(word): Returns the related words, ancestor and descendant information of a given word.

Class name	LanguageSea
Attributes	languageFamilies: The language families that are currently included in the language sea.
Functions	addLanguageFamily(language): Adds a new language to currently displayed language sea. getLanguageFamilies(): Returns the language families.

Class name	Word
Attributes	definition: Dictionary definition of the word. context: Context of the word. features: Certain features extracted from the word for machine learning purposes. languageFamily: Language family which the word is part of.
Functions	getWord(): Returns the word itself getRelatedWords(): Returns the related words for the specific word. getAncestor(): Returns the ancestor word of the specific word.

Class name	EtymonController
Attributes	languageSea: The language sea consists of all the languages and words that are connected to each other.
Functions	startEtymon(): Initiates the program, connects to databases searchWord(Word): Queries a word search in Etymon database and displays the results trainMLModule(): Initiates the clustering algorithm on database hallucinate(Word): Creates new words hallucinateML(Word, List): Runs the machine learning in hallucination mode, using generative LSTM. getRandomWord(): Chooses and displays a random word from the Etymon database. generateLanguageSea(lang1, lang2...): Generates and displays a language sea with the given languages. generateWord(Word): Runs the new word through the machine learning algorithm. generateWordCloud(Word): The newly added word is run through the ML algorithm to generate connection with other words and form a cloud. scanObject(): Scans the object using camera and returns its name. Will be used to search a word using the object recognition algorithm. zoominAR(): Zooms the image in AR. generateWordCloudAR(): Generates and shows the

	word cloud of a specific word as AR.

Class name	WebGL
Attributes	graphics: The actual graphics to display (language see, zoomed-in word cloud, etc.)
Functions	renderGraphics(): Draws the language sea on display. zoomIn(Word): Finds and zooms in the the queried word. zoomOut(): Display returns to the initial state. addWord(Word): User requests a new word to be added to the Etymon database. displayLanguageSea(): Displays the language sea. displayLanguageSea(lang1, lang2...): User chooses which languages will be displayed in the language sea. hallucinate(): generates the graphics for hallucination mode.

Class name	GraphicsController
Functions	getLanguageSea(): Gets the language sea from Etymon database to be used in WebGL. getLanguageSea(lang1, lang2...): Retrieves some of the languages that will be displayed in the language sea. getWord(word): Returns the word data on a scpecific word. getWordCloud(): Gets the word cloud data from database. getHallucination(): Retrieves the hallucination data.

4. Machine Learning Model Design

In this section four different algorithms that can be used as the machine learning components of Etymon will be discussed. Each of their advantages and disadvantages will also be analyzed. Logistic analyses of all the models is also included to decide on which algorithm to deliver in the end product.

4.1. Deep Learning Approach

Using Deep Neural Networks (DNN), ancestor words of current words from different languages of today can be found. This is a generative model, as it learns the training information and then generates a learning method by itself to be applied to other cases. In the end, this can create proto-words—ancestor words—can be created using Long Short-Term Memory (LSTM) models.

In the training part of the DNN, labels will be ancient words that are already known and the training set will consist of their descendant words in several different languages. All of the word data will be downloaded from wiktionary Proto-Indo-European word list [4].

The main advantage of this model is that it creates a simple implementation of the machine learning part. Furthermore, no manual feature extraction is necessary with this approach, as the words would be provided to the DNN and the output will be received after the training and testing periods. The main disadvantages are that the data classes are unbalanced and there is a small amount of data, which may reduce the accuracy of the DNN. Even more, the training period would be long for a DNN.

4.2. Graph Alignment Approach

Recent research in applied graph theory is focusing on graph alignments of different social media to identify missing components in another graph. This same concept can be used in identifying missing words in different languages and in identifying ancestor words.

Using wiktionary data for nearly five million words in the English section, graphs can be generated for multiple languages. Edges in the node will be connecting different words in the same language with which the current node has a link in the wiktionary page. Then the machine learning model can look at links between two graphs after alignment in order to see the word resemblances

Even though this model can identify the links between words of different languages, it does not fully create a proto-word, which would be a shortcoming as Etymon is also responsible for giving root information. Furthermore, the wiktionary data may not depict correct linguistic links in all of its word entries

4.3. Dynamic Programming Approach

In regular linguistic epigenetics, a method called comparative method is utilized to find the ancestry information of a language. This method is generally automated by using a dynamic programming strategy. This approach also borrows from the bioinformatics community by using the "edit distance" solution of dynamic programming [5]. With this approach, two words in two languages can be compared at the same time with one another. One language can also be a proto-language. Hence, a word's edit distance can be found from its proto-word to a real word in today's languages. Using this strategy, we will be generating multitudes of dynamic programming tables. Hence, we can use the patterns found in these tables as machine learning features.

One drawback of this approach can be that it can only work with borrowed words in different languages, or it can favor borrowed words than any other word as the characters will have very small edit distances. Furthermore, every new word would be needed to be compared with every other word in the other language.

4.4. Word Embeddings Approach

Like with the graph alignment strategy, each word can be clustered according to their meanings and can be transformed into a vector using a tool like word2vec [6]. This would create a hyperdimensional space, and these spaces can be created for two different languages. In the end, the differences between these two spaces can be used to predict the ancestor words.

The main advantage of this method is that it successfully transforms words into vectors and multilingual forms of the word2vec can be employed.

All of these approaches are possible, yet doing them all at the same time would require a lot of time that can exceed the delivery time of the project. As a result, currently the most feasible approach—pragmatic logistics-wise—is the deep learning approach.

5. References

[1] E. P. Hamp and J. Lyons, "Linguistics," Encyclopædia Britannica, 10-Mar-2017. [Online]. Available: https://www.britannica.com/science/linguistics/The-comparative-method. [Accessed: 12-Feb-2018].

[2] "About," Ethnologue. [Online]. Available: https://www.ethnologue.com/about. [Accessed: 12-Feb-2018].

[3] C. Diagne and N. Barradeau, *Free Fall,* https://artsexperiments.withgoogle.com/freefall/wave. [Accessed: 09-Oct-2017].

[4] "Appendix:List of Proto-Indo-European roots," Appendix:List of Proto-Indo-European roots - Wiktionary. [Online]. Available: https://en.wiktionary.org/wiki/Appendix:List_of_Proto-Indo-European_roots. [Accessed: 12-Feb-2018].

[5] M. P. Oakes, "Computer Estimation of Vocabulary in a Protolanguage from Word Lists in Four Daughter Languages," Journal of Quantitative Linguistics, vol. 7, no. 3, pp. 233–243, Jan. 2000.

[6] Dav, "dav/word2vec," GitHub, 18-Jan-2018. [Online]. Available: https://github.com/dav/word2vec.git. [Accessed: 12-Feb-2018].