Université Ibn-Tofail
Faculté des Sciences
Département d'Informatique

Licence Fondamentale en Sciences Mathématiques et Informatiques

Projet de Fin d'Etudes

Mémoire présenté par

Aarab Abderrahmane

Sous le thème

Arabic Natural Language Processing: Application for Gerunds in Arabic Language

Présenté devant le Jury composé de :

Pr Mohammed Mourchid, Faculté des Sciences de Kénitra, *Encadrant*

Pr. Hatim Kharraz Aroussi, Faculté des Sciences de Kénitra, *Examinateur*

Final Year Project Report

Arabic Natural Language Processing:

Application for Gerunds in Arabic Language

Abderrahmane Aarab

A report submitted in partial fulfillment of the degree of

BSc in Mathematical & Computer Sciences (SMI)

Supervisor: Prof**.** Mohammed Mourchid

Department of Computer Science
Ibn Tofail University, Kenitra

Declaration & Acknowledgments

This project report was submitted to fulfill the Bachelor of Science in Mathematical & Computer Sciences at the Department of Computer Science at Ibn Tofail University. This study has been under the supervision of Prof. Mohammed Mourchid, Senior Lecturer, Ibn Tofail University at the Department of Computer Science.

I would like to express my sincere thanks to the people who helped me and contributed to the elaboration of this work as well as to the success of this wonderful academic year. I would like to also express my deepest gratitude to Mr. Mourchid Mohamed, who, as a supervisor, has always shown himself to be a good listener and very available throughout the realization of this project, as well as for the inspiration, the help, and the time that he has kindly devoted to me and my peers. My thanks are also addressed to Ms. Asmaa Amzali and Ms. Asmaa Kourtin, for the patience they have shown towards us and the precious help they have given us throughout this project period. I also present my anticipated thanks to the members of the jury who gave us the privilege of accepting to evaluate our work. My sincere thanks also go to all the teachers of the computer science department of the Faculty of Science of Ibn Tofail for having put so much energy to ensure us a quality education and training.

Student Name: Abderrahmane Aarab

Date of Submission: 27/06/2022

Signature:

Abstract

This project presents a JAVA application for treating Gerunds in the Arabic Language, which contributes to facilitating the learning process of the Arabic language. It uses NooJ linguistic platform to analyze the given text or file. NooJ's linguistic engine with its Text Annotation Structure (TAS) returns an annotation file after doing the linguistic analysis. The application stands on this returned annotation file to recognize and represent both verbs and gerunds that occur in the text. We assume that the user must be able to distinguish between nouns, verbs, and other grammatical categories, but they are not able to extract more sophisticated linguistic features, e.g. the pattern of special verbs or gerunds that have a duplicated root. The difficulty lies in the changes that occur in these kinds of words. The application is meant to recognize and represent these sophisticated linguistic features and provide them to the user. The representation process provides the user with the linguistic features of any verbs and gerunds that occurred in the inputted text or file, e.g. the application would be able to return root, singular pattern, type, form, transitivity, conjugation info, grammatical category, and other derivational and inflectional features of a certain verb or gerund. The application can also examine rules that were used to generate gerund forms from its verb, depending on the type of verb. By making a powerful and easy-to-use interface, the java application aims to make the user capable of extracting all possible gerund forms and verbs in any inputted text or file, converting a verb to its different gerund forms (& reverse), and transforming an Arabic verb phrase into a noun phrase, while also providing the original form and linguistic info of a verb or gerund.

Keywords: Natural Language Processing **-** Arabic Natural Language Processing **-** Arabic learning system - Arabic verbs - Arabic gerunds - Inflections – Derivations - Arabic morphology - NooJ platform - Linguistic features

Table of Contents

List of Figures

List of Tables

General Introduction

The Arabic language is a Semitic language that is very rich in morphological phenomena (i.e. inflectional and derivational). It contributes to the construction of an extensional lexicon with wide coverage and guarantees the reusability of the resources, mostly using a unification grammar. This kind of formalism offers complete representation with a minimum number of rules. Among the most complicated derivational forms, we find the gerund for Arabic nouns. However, works treating Arabic gerunds are very limited or almost non-existent. Indeed, gerund has several forms and its treatment is not evident. It is based on several criteria (i.e., the type, the scheme of the verb, and the semantic aspect), which makes it difficult to find the hierarchy type representing the proposed patterns. In this context, we are interested in transforming a verb phrase into a noun phrase, generating the gerund of a given verb, and getting a verb from a given gerund. To do this, we begin by studying the Arabic gerund to generate its different characteristics. Based on this study, we identified the different constraints characterizing each type of Arabic gerund. These constraints were described first in an Excel file with their corresponding verbs including a set of features. The originality of this work appears in the use of the NooJ linguistic platform to model dictionaries and grammars, which are defined by derivational, inflectional, morphological, and syntactic graphs. In addition, relying on the object-oriented paradigm to specify the hierarchy and relation of types of gerund and verb shows an interaction between computer science and linguistics. Moreover, the lack of researchers treating the Arabic derivations and inflections especially gerund forms with Nooj represents another novelty. In the present report, we begin by presenting a detailed linguistic study about the Arabic verbs and gerunds. According to this study, we present in the next sections the language resources, the constructed Nooj dictionary, derivational/inflectional grammars for Arabic gerunds, and their corresponding verbs with their linguistics specifications. Then, we experiment and evaluate the different verb and gerund forms with Nooj. In addition to that, we display how our Java application would be linked to Nooj so that it could use NooJ's features like syntactic analysis (Text Annotation). After a detailed view of the Java application, we finally end with a conclusion and some perspectives.

Definitions

1.1 Natural Language

A natural language or ordinary language is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation. Natural languages can take different forms, such as speech or signings like Arabic, French, and English.

1.2 Arabic Natural language

Arabic is a Semitic language spoken by more than 330 million people as a native language, in an area extending from the Arabian/Persian Gulf in the East to the Atlantic Ocean in the West. Arabic is a highly structured and derivational language where morphology plays a very important role. Morphology is central in working on Arabic NLP because of its important interactions with both orthography and syntax. Arabic's rich morphology is perhaps the most studied and written about aspect of Arabic. As a result, there is a wealth of terminology, some of it is inconsistent and may intimidate and confuse new researchers.

1.3 Natural language processing

Natural language processing is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.

1.4 Arabic Natural language processing

Over the last decade, Arabic and its dialects have begun to gain ground in the area of research within Natural Language Processing (NLP). Much work targeted different aspects related to how this language and its dialects are processed, such as Morphological analysis, resource building, Machine translation, etc. To present the characteristics of this language and to classify the works handling it, different surveys were proposed. Morphology is central in working on Arabic NLP because of its important interactions with both orthography and syntax. Arabic's rich morphology is perhaps the most studied and written about aspect of Arabic. As a result, there is a wealth of terminology and many inconsistencies.

1.5 NooJ Platform

NooJ is a linguistic developmental environment, which can analyze texts of several million words in real-time. It includes tools to construct, test, and maintain large coverage of lexical resources, as well as morphological and syntactic grammars. Dictionaries and grammars are applied to texts to locate derivational, inflectional, morphological, and syntactic patterns, remove ambiguities, and tag simple and compound words. NooJ platform works on a cascade model; the result of each analysis step is the input of the next one. For more information, please consult the official NooJ website.

We adopted this platform because it allows us to:

- Implement all linguistic analysis phases: lexical, morphological, syntactical, and semantic analyses.
- Create our dictionary and apply the search option using special queries.
- Implement our grammars and dictionary using its linguistic engine.
- Analyze our text by giving derivational, inflectional, and syntactical properties of each word/sentence.

1.6 NooJ Architecture

NooJ platform is Programmed using C#/.net Framework. NooJ follows a component-based software approach, which is a step beyond the object-oriented programming paradigm. The system consists of three modules, corpus handling, lexicon and grammar development that are integrated into a single intuitive graphical user interface (command line operation is also available).

NooJ processes texts and corpora (i.e. sets of text files) at the Orthographical, Morphological, Syntactic, and Semantic levels. All linguistic information (at any level) is represented by annotations that are stored in the Text Annotation Structure (TAS).

Using this platform allows us to formalize the Arabic 3-lettered or 4-lettered verb model as a first step of the Arabic dictionary constructing phase. Starting by building our dictionary that contains the previous verb category and linking it with our productive grammars that give all derivational and inflectional forms for each dictionary entry, we will detail this in the next sections.

Chapter 1: **Linguistic Study on Arabic Language**

We present in this chapter a quick linguistic study on the Arabic language, mainly about verbs, gerunds, and the subjunctive case in Arabic. This linguistic study helps give an overview of the Arabic language's structure and provides us with a solution for the upcoming tasks including:

- Establishing a relation between verbs and gerunds.
- Converting a given verb to its different types of the gerund.
- Converting a given gerund to its original verb.
- Converting a given Arabic verb phrase to a noun phrase.

1.1 Arabic Verbs

Arabic verbs, like the verbs in other Semitic languages, and the entire vocabulary in those languages, are based on a set of two to five (but usually three) consonants called a root (trilateral or quadrilateral according to the number of consonants). The root communicates the basic meaning of the verb, e.g. ك-ت-ب k-t-b 'write', ق-ر-ء q-r-aʾ 'read', ء-ك-ل ʾa-k-l 'eat'. Changes to the vowels in between the consonants, along with prefixes or suffixes, specify grammatical functions such as the person, gender, number, tense, mood, and voice. Various categories are marked on verbs:

- Three tenses (present, past, and future tense, where the last one is indicated by the prefix sa- or the particle sawfa and the present tense).
- Two voices (active, passive).
- Two genders (masculine, feminine).
- Three persons (first, second, third).
- Three numbers (singular, dual, plural).

- Six moods in the non-past only (indicative, subjunctive, jussive, imperative, and short and long energetics).

- Nineteen forms, the derivational systems indicating derivative concepts such as intensive, causative, reciprocal, reflexive, frequentative… etc. For each form, there is also an active and a passive participle (both adjectives, declined through the full paradigm of gender, number, case, and state) and a verbal noun (declined for case; also, when lexicalized, may be declined for number).

Arabic grammarians typically use the root ف-ع-ل *f-aa-l* to indicate the particular shape of any given element of a verbal paradigm, also called the pattern. For example, the form يتكاتب (root: ك-ت-ب) *yutakātab corresponds* with the pattern that would be listed generically as يتفاعل *yutafāʿalu*, specifying the generic shape of a strong Form VI passive verb, third-person masculine singular present indicative.

The maximum possible total number of verb forms derivable from a root — not counting participles and verbal nouns — is approximately 13 person/number/gender forms; times 9 tense/mood combinations, counting the س- *sa-* future (since the moods are active only in the present tense, and the imperative has only 5 of the 13 paradigmatic forms); times 17 form/voice combinations (since forms IX, XI–XV exist only for a small number of stative roots, and form VII cannot normally form a passive), for a total of 1,989. Each of these has its own stem form, and each of these stem forms itself comes in numerous varieties, according to the weakness (or lack thereof) of the underlying root.

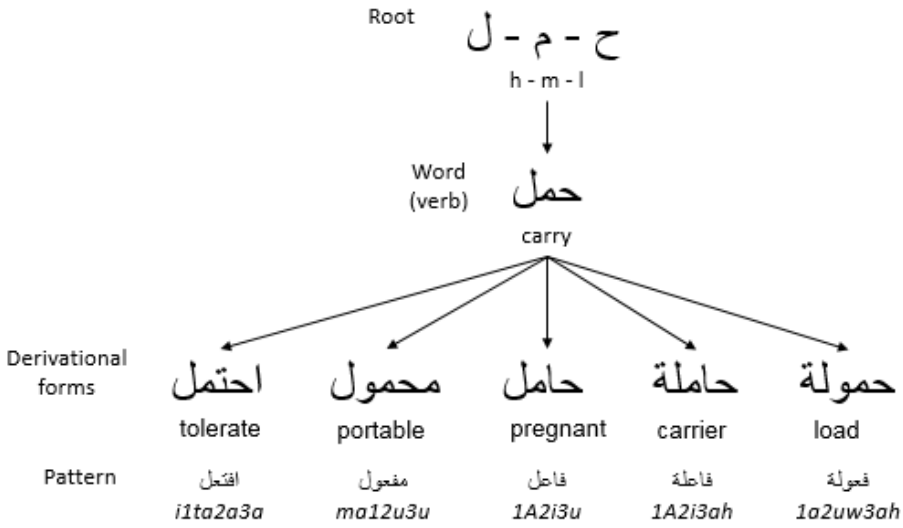

Figure 1 Derivations of The Arabic Verb (carry) & Their Patterns

1.2 Arabic Gerunds

In addition to a participle, there is a verbal noun (in Arabic, مَصْدَر *maṣdar*, pl. مَصَادِر *maṣādir*, literally meaning 'source'), sometimes called a gerund, which is similar to English gerunds and verb-derived nouns of various sorts (e.g. "running" and "a run" from "to run"; "objection" from "to object"). As shown by the English examples, its meaning refers both to the act of doing something and (by frequent semantic extension) to its result. One of its syntactic functions is as a verbal complement of another verb, and this usage it corresponds to the English gerund or infinitive (*He prevented me from **running*** or *He began **to run***).

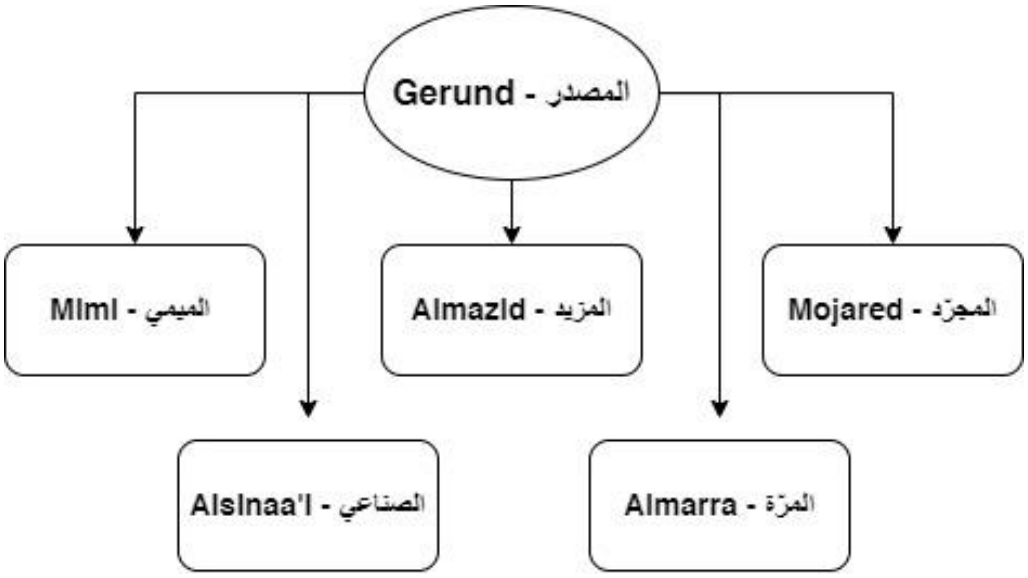

Figure 2 Gerund Type Hierarchy.

The gerund is a derivational noun obtained from an Arabic verb. It expresses the principal idea of an action or an action that has no reference time. As represented in Figure 2, we see the five sub-types tackled in this project.

Each sub-type of gerund, illustrated in Figure 2, has specific criteria representing the base for its construction. First, the Gerund-*Mojared* "مصدر مجرد" can be obtained from an un-augmented verb (فعل مجرد */ fi'l mojared*). In addition, this type of Gerund has several types of forms (i.e. regular and irregular).

To construct regular forms, for example, we are based on the pattern of a verb by adding semantics or transitivity (i.e. transitive or intransitive) descriptions. Table 1 illustrates some regular forms of Gerund-*Mojared.*

Table 1 Regular Forms of Gerund-*Mojared.*

Verb Pattern	Semantic / Transitive		Examples
فَعَلَ (fa'ala)	Semantic	A trade meaning	زرع –> زِرَاعةٌ to plant-> the planting
		A voice meaning	صرخ–> صراخ to scream-> the screaming
	Transitive	Intransitive	جلس –> جلوس to sit-> the sitting

As shown in Table 1, for example, if an Arabic verb has the scheme *fa'ala* and its semantic description is a voice meaning, the gerund-*Mojared* is obtained according the pattern *fu'oul* which adds the letter "w/ و" before the last letter of the verb. While, the irregular forms for the gerund- *Mojared* can have several patterns either by adding one or more letters such as "(و,w) / (ى,y) / (ة,t) / (ا, A) ", either by modification of vocalization, or by the combination of two cases of change.

For example, the two verbs (فتح <to open> / زرع <to plant>) have the same scheme (*fa'ala_yaf'alou* فعل_يفعَلُ) and the same type "Intact" but we can't apply the same rule to obtain the Gerund-*Mojared* فَتْحٌ) <the opening> / زِرَاعة <the planting>). In fact, the first forms of gerund-Mojarad (i.e. فَتْحٌ <the opening>) is an irregular form obtained by modifying

the vocalization. However, most of gerund-*Almazid* "مصدرالمزيد " forms can be obtained with regular method applied to an augmented Arabic verb (فعل مزيد/ fi'l mazid). In fact, this type has 15 patterns according to the pattern of verb.

By the way, for the gerund-*Mimi*" المصدر الميمي " we add generally, the prefix " م" to an Arabic verb. If the verb has three letters, the vocalization of the prefix will be either "م /*ma*" or either " مُ/*mu*".

For the gerund-*Alsinaa'I "*المصدر الصناعي*",* a noun can be obtained from a participle noun or active noun or superlative noun or proper noun or another type of gerund such as the gerund-*Mimi*. In fact, for each type of this noun, we add the sign of the feminine, the letter "ة/t". For example, the active participle "عالم /Scientist/'aalimoun" will be "عالميّة /International/'aalamiyatun" for the gerund-*Alsinaa'i*.

The Gerund-*Almarra* "مصدر المرّة" means that the verb is applied only once. In fact, if the verb is composed by three letters, this type of gerund became transformed to the schema "فعلة / fi'latun". Otherwise, this type became transformed according to the scheme of its verb with adding the letter "ة/ t" at the end. According to this linguistic study, we conclude that each sub-type of gerund has specific patterns.

1.3 The subjunctive case with "ان" in Arabic

The subjunctive as a concept in grammar refers, in general, to that which is uncertain or related to emotion. Often it is used for things which are sought, desired, or feared, but which are not necessarily realized. In such situation, verbs in many languages reflect the somewhat uncertain nature of what is happening through changes in their conjugations. In many languages, Spanish for example, the subjunctive exists in more than one tense and the conjugations can become rather confusing. In Arabic, however, the subjunctive is only used as a mood of the imperfect tense. As a result, there is only one way to conjugate a verb in the subjunctive in Arabic.

The subjunctive in Arabic occurs in situations. One is after particles such as لِ , كَيْ and لِكَيْ which essentially mean the same thing: “in order to” or after the particle لَنْ which is used to negate the future. The other situation is after the word أنْ , which is used with verbs of desire, emotion, or intention. In this project, we will deal with the three situation. Regardless of which of the two uses of the subjunctive is being employed, the subjunctive conjugations will be the same.

Below are the subjunctive conjugations for the verb يَدْرُسُ, دَرَس

Table 2 Subjunctive Conjugations For The Verb يَدْرُسُ, دَرَس

Plural		Dual		Singular	
نَدْرُسَ	نَحْنَ	تَدْرُسَا	أنتُما	أدْرُسَ	أنا
تَدْرُسُوا	أنتُم	يَدْرُسَا	هما (m)	تَدْرُسَ	أنتَ
تَدْرُسْنَ	أنتُنَّ	تَدْرُسَا	هما (f)	تَدْرُسِي	أنتِ
يَدْرُسُوا	هم			يَدْرُسَ	هو
يَدْرُسْنَ	هنَّ			تَدْرُسَ	هي

As you can see, the differences between the subjunctive conjugations and the present tense conjugations are minimal. For the “big five” (whenever the present tense suffix is a dhamma), the subjunctive suffix is a fatha. For أنتِ , we drop the final نَ just as we do for the jussive. Whenever the present tense suffix is ونَ the subjunctive is وا with the alif unpronounced. For the second and third person feminine plurals the conjugations are the same as they are for the present tense and the jussive.

A simple way to look at it is this: All five present tense conjugations, which end in a dhamma end instead with a fatha in the subjunctive. For all other conjugations in the subjunctive, you use the jussive conjugations.

Using this linguistic analysis on simple Arabic 3-lettered verbs should be a great reference when dealing with more complex verbs and word forms.

Chapter 2: **NooJ & Language Resources**

There are many phases involved in natural language processing: Lexical, Morphological, Syntactic, and Semantic Analyses. We briefly define some of these steps.

1. **Lexical Analysis:** In computer science, lexical analysis, lexing or tokenization is the process of converting a sequence of characters into a sequence of lexical tokens. A program that performs lexical analysis may be termed a lexer, tokenizer, or scanner, although scanner is also a term for the first stage of a lexer.

2. **Morphological Analysis:** This involves dividing a text into paragraphs, words and the sentences and its main role is to represent the Atomic Language Unit (ALUs) which is the smallest elements that make up the sentence, we are going to define these ALUs/3 lettered verbs as dictionary entries that represent the language vocabulary

3. **Syntactic Analysis:** This involves analyzation of the words in a sentence to depict the grammatical structure of the sentence. The words are transformed into structure that shows how the words are related to each other E.g. “The girl the go to the school”. This would definitely be rejected by the English syntactic analyzer.

4. **Semantic Analysis:** This abstracts the dictionary meaning or the exact meaning from context. The structures, which are created by the syntactic analyzer, are assigned meaning. There is a mapping between the syntactic structures and the objects in task domain. E.g. “Colorless blue idea”. The analyzer would reject this, as colorless blue do not make any sense together.

2.1 Language Resources

The most important thing during building any language recourse is to ensure that it reflects the structure of this language. For this reason, all the Arabic resources were built using root and pattern approach. The components and resources required are illustrated in Figure 3. The first one is a fully Arabic verb model based on root and pattern approach that contains all Arabic verbs models with their inflectional and derivational forms.

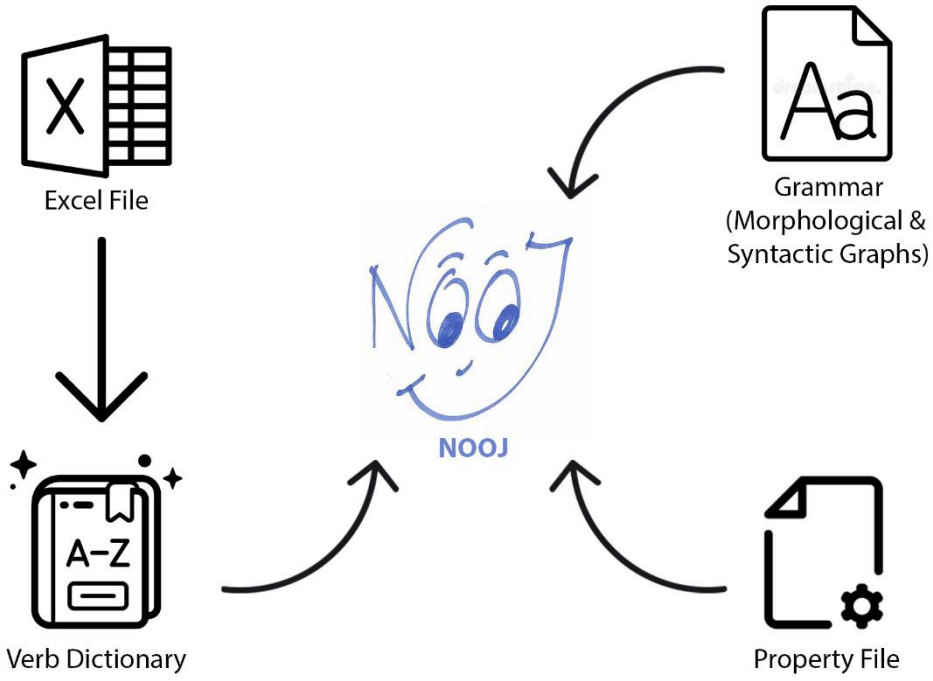

Figure 3 Arabic Linguistic Resources for NooJ

- **The Excel File:** This contains a collection of Arabic verbs, their semantics, and corresponding gerunds.

- **The dictionary:** This is generated from the excel file. It contains verbs linked to their morphological and semantic features; they are also linked to their inflectional and derivational forms. Due to time constraints, the dictionary contains only the Arabic verbs that start with the letter "(R/ر)". However, the concept still applies to every other verb in the Arabic language.

- **Grammars:** This contain all derivational and inflectional forms of any dictionary entry.

- **Property file:** This file plays the same rule as variables declaration play in any programming language.

2.1.1 Arabic 3-Lettered/Trilateral Verbs

Most Arabic words are derived from three-letter Verbs or 3- lettered verbs. As each Arabic verb has its morphological properties like root and pattern, we attached each verb/dictionary entry with this properties, and with their conjugation form. For instance, the verb (to link – رَبَطَ - RaBaTa) takes "ر ب ط" as roots letters (in Arabic the root letters are separated *fa'ala* - فَعَلَ as pattern and *yaf'alou* - يفعَلُ as conjugation form . The conjugation form of the previous verb: رَبَطَ - RaBaTa is يَرْبَطُ yaRBaTo according to the matching process between the pattern and the conjugational form *fa'ala*- فَعَلَ_ *yaf'alou*-يفعَلُ. There are three types of the 3-lettered verbs patterns in Arabic. They are distinguished according to the second letter vowel:

(fatha aَ kasra iِ , dama uُ).

- فَعَلَ - faEala is a pattern of the verb: (to link - رَبَطَ - RaBaTa).
- فَعُلَ - faEula is a pattern of the verb : (to tweak - رَبُجَ - RaBuJa).
- فَعِلَ - faEila is a pattern of :(to shove - رَبِشَ - RaBiSha).

These patterns may take three conjugation forms as table 2 shows.

Table 3 All possible patterns with their conjugation forms in 3-letter Arabic verbs

Verb Pattern	Conjugation Form 1	Conjugation Form 2	Conjugation Form 3
faEala - فَعَلَ	yafEalo - يَفْعَلُ	yafEilo - يَفْعِلُ	yafEolo - يَفْعُلُ
faEula - فَعُلَ	yafEalo - يَفْعَلُ	yafEilo - يَفْعِلُ	yafEolo - يَفْعُلُ
faEila - فَعِلَ	yafEalo - يَفْعَلُ	yafEilo - يَفْعِلُ	yafEolo - يَفْعُلُ

2.1.2 Data Collection / Excel File

Data collection is a crucial part in NLP, in our case we have to collect Arabic Verbs that start with the letter "(R/ر)", their root, root info, pattern, syntactic form, transitivity, conjugation info, the corresponding gerunds, and other features. The collection of Arabic verbs and gerunds that start with the letter "(R/ر)" was done through the use of the dictionary Al-Muʿjam Al-Wasīṭ "المعجم الوسيط" , as shown in Figure 4.

verb	RT	TYP	TRAN	FRM	PAT	RI	CI	FLX	MSD	MSD-PAT
رَأَبَ	رأب	Tri	Tr	VN0N1	فَعَلَ	CHC	aa	FLX=aaCHC	رَأباً	فَعْل
رَأَدَ	رأد	Tri	Intr	VN0	فَعَلَ	CHC	aa	FLX=aaCHC	رَأداً	فَعْل
رَؤُدَ	رأد	Tri	Intr	VN0	فَعُلَ	CHC	uu	FLX=uuCHC	رُؤُدَةً	فُعُولَة
ارْتَادَ	رأد	Qua	Intr	VN0	افْتَعَلَ	CHC		FLX=افْتَعَلCHC	ارْتِيَادٌ	افْتِعَال
تَرَاءَدَ	رأد	Qua	Intr	VN0	تَفَاعَلَ	CHC		FLX=تَفَاعَلCHC	تَرَاؤُدٌ	تَفَاعُل
تَرَأَّدَ	رأد	Qua	Intr	VN0	تَفَعَّلَ	CHC		FLX=تَفَعَّلCHC	تَرَئِيد	تَفْعِيل
رَأْرَأَ	رأرأ	Qua	Intr	VN0	فَعْلَلَ	HDD		FLX=فَعْلَلHDD	رَأْرَأَةٌ	فَعْلَلَة
رَأَسَ	رأس	Tri	Tr	VN0N1	فَعَلَ	CHC	aa	FLX=aaCHC	رَأسَةً-رَأْسًا	فَعَالَة-فَعْل
رَئِسَ	رأس	Tri	Intr	VN0	فَعِلَ	CHC	ia	FLX=iaCHC	رَأساً	فَعَل
رَاءَسَ	رأس	Qua	Intr	VN0	فَاعَلَ	CHC		FLX=فَاعَلCHC	مُرَاءَسَة	مُفَاعَلَة
ارْتَأَسَ	رأس	Qua	Intr	VN0	افْتَعَلَ	CHC		FLX=افْتَعَلCHC	ارْتِئَاسٌ	افْتِعَال
تَرَأَّسَ	رأس	Qua	Intr	VN0	تَفَعَّلَ	CHC		FLX=تَفَعَّلCHC	تَرَئِيسٌ	تَفْعِيل
رَأَفَ	رأف	Tri	Intr	VN0	فَعَلَ	CHC	aa	FLX=aaCHC	رَأفاً	فَعَل
رَئِفَ	رأف	Tri	Intr	VN0	فَعِلَ	CHC	ia	FLX=iaCHC	رَأفًا	فَعَل
رَؤُفَ	رأف	Tri	Intr	VN0	فَعُلَ	CHC	uu	FLX=uuCHC	رَآفَةً	فَعَالَة

Figure 4 Verbs & their Gerunds in Excel File

- **Verb:** This represents the verb as an entry. [رَأَبَ]
- **RT:** This is the root of the verb. [ر-أ-ب]
- **TYP:** This represents the type of the verb (Tri=Trilateral / 3 lettered). [Tri]
- **TRAN:** This dictates if the verb is transitive or intransitive. [Tr = Transitive]
- **FRM:** This is the syntactic form of the verb in a phrase. [VN0N1 = Verb Noun0 Noun1]
- **PAT:** This is the pattern of the verb. [فَعَلَ]
- **RI:** This is a general form of the root describing the letters. [CHC = Consonant-Hamza-Consonant]
- **CI:** This is a representation of the changing vowel of the verb in the past and the present tense. [aa]
- **MSD: (**Massdar-*Mojared Assli*)**:** This is the original gerund of the verb. [رَأْب]

Conclusion: Now that we have a collection of all the required data, to facilitate the next steps (Creation of the Nooj Dictionary & Grammars) we add the names of the derivational graphs (DRV) and flectional graphs (FLX) of different types of gerunds for each verb.

2.2 Lexical Analysis using NooJ

The first level of text analysis requires the computer to identify the **Atomic Linguistic Units** (ALUs) of the text, i.e. its smallest non-analyzable elements. In the next sections, we define these ALUs, and we show how NooJ's dictionaries are used to describe them. We also describe NooJ's inflectional and derivational grammars. Finally, we present NooJApply, NooJ's API, which uses dictionaries, the inflectional and derivational engine, and syntactical grammars to annotate words in a given text.

2.2.1 NooJ Dictionary

NooJ dictionaries are used to represent, describe and recognize simple words, multi-word units as well as discontinuous expressions. The dictionaries displayed here are ".nod" files that have been compiled from editable ".dic" source files. Technically, ".nod" files are finite-state transducers; we speak of them as compiled dictionaries because, from the point of view of the end user, they come out of editable text ".dic" type documents that were compiled using the **Lab > Dictionary > Compile** command.

The lexicon of a language is its vocabulary that includes its words and expressions, while morphological Analysis involves dividing a text into paragraphs, words and the sentences and its main role is to represent the Atomic Language Unit (ALUs), we are going to define these ALUs/3-lettered verbs as dictionary entries that represent the language vocabulary. These entries are associated with their morphological properties which enrich it with linguistic information like: (s) means singular, (p) means plural, as basic properties while we add our specific verb grammatical, lexical, morphological, syntactical properties like: (Root/Pattern/Transitivity…etc) that will be used in advanced analysis phases.

Figure 5 shows our constructed dictionary that calls at first our inflectional and derivational grammars (grammar.nof), as it is shown in figure 5, the dictionary contains the language vocabulary with their special morphological properties: V: verb, Tr : transitive verb , root (رأب), pattern (faEala - فَعَلَ).

```
# NooJ V7
# Dictionary
#
# Language is: ar
#
# Alphabetical order is not required.
#
# Use inflectional & derivational paradigms' description files (.nof), e.g.:
# Special Command: #use paradigms.nof
#
# Special Features: +NW (non-word) +FXC (frozen expression component) +UNAMB (unambiguous lexical entry)
#                   +FLX= (inflectional paradigm) +DRV= (derivational paradigm)
#
# Special Characters: '\' '"' ' ' ' ' '+' '-' '#'
#use G_grammar.nof
#use V_grammar.nof

رَأَبَ,V+TYP=Tri+RT=رأب+TR=Tr+FRM=VN0N1+RI=CHC+PT=فَعَلَ+CI=aa+FLX=aaCHC+DRV=DCHC:فَعَل;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiD
رَادَ,V+TYP=Tri+RT=رأد+TR=Intr+FRM=VN0+RI=CHC+PT=فَعَلَ+CI=aa+FLX=aaCHC+DRV=DCHC:فَعَل;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiDCC
رَؤُدَ,V+TYP=Tri+RT=رأد+TR=Intr+FRM=VN0+RI=CHC+PT=فَعُلَ+CI=uu+FLX=uuCHC+DRV=DCHC:فُعُولَة;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiDC
اِرْتَأَدَ,V+TYP=Qua+RT=رأد+TR=Intr+FRM=VN0+RI=CHC+PT=اِفْتَعَلَ++DRV=DCHC:اِفْتِعَال;FLXDRV+DRV=SDCCC:يَة;FLXDRV
تَرَاءَدَ,V+TYP=Qua+RT=رأد+TR=Intr+FRM=VN0+RI=CHC+PT=تَفَاعَلَ++DRV=DCHC:تَفَاعُل;FLXDRV+DRV=DCHC:تَفَاعَلَ;MaFLXة+DRV=MiDCCC:مُتَفَاعِل;FLXDRV+DRV=SDCCC:يَة;FLXDRV
تَوَرَّدَ,V+TYP=Qua+RT=ورد+TR=Intr+FRM=VN0+RI=CHC+PT=تَفَعَّلَ++DRV=DCHC:تَفْعِيل;FLXDRV+DRV=DCHC:تَفَعَّلَ;MaFLXة+DRV=MiDCCC:مُتَفَعِّل;FLXDRV+DRV=SDCCC:يَة;FLXDRV
رَأْرَأَ,V+TYP=Qua+RT=رأرأ+TR=Intr+FRM=VN0+RI=HDD+PT=فَعْلَلَ++DRV=DHDD:فَعْلَلَة;FLXDRV+DRV=DHDD:فَعْلَلَ;MaFLXة+DRV=MiDCCC:مُفَعْلِل;FLXDRV+DRV=SDHDD:يَة;FLXDRV
رَأْسَ,V+TYP=Tri+RT=رأس+TR=Tr+FRM=VN0N1+RI=CHC+PT=فَعَلَ+CI=aa+FLX=aaCHC+DRV=DCHC:فَعَل;FLXDRV+DRV=DCHC:فَعْلَة;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDC
رَئِمَ,V+TYP=Tri+RT=رأس+TR=Intr+FRM=VN0+RI=CHC+PT=فَعِلَ+CI=ia+FLX=iaCHC+DRV=DCHC:فَعَل;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiDCC
رَاءَسَ,V+TYP=Qua+RT=رأس+TR=Intr+FRM=VN0+RI=CHC+PT=فَاعَلَ++DRV=DCHC:مُفَاعَلَة;FLXDRV+DRV=DCHC:فَاعَلَ;MaFLXواحدة+DRV=MiDCCC:مُفَاعِل;FLXDRV+DRV=SDCCC:يَة;FLXDR
اِرْتَأَسَ,V+TYP=Qua+RT=رأس+TR=Intr+FRM=VN0+RI=CHC+PT=اِفْتَعَلَ++DRV=DCHC:اِفْتِعَال;FLXDRV+DRV=SDCCC:يَة;FLXDRV
تَرَأَّسَ,V+TYP=Qua+RT=رأس+TR=Intr+FRM=VN0+RI=CHC+PT=تَفَعَّلَ++DRV=DCHC:تَفْعِيل;FLXDRV+DRV=DCHC:تَفَعَّلَ;MaFLXة+DRV=MiDCCC:مُتَفَعِّل;FLXDRV+DRV=SDCCC:يَة;FLXDRV
رَأَفَ,V+TYP=Tri+RT=رأف+TR=Intr+FRM=VN0+RI=CHC+PT=فَعِلَ+CI=aa+FLX=aaCHC+DRV=DCHC:فَعَل;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiDC
رَئِفَ,V+TYP=Tri+RT=رأف+TR=Intr+FRM=VN0+RI=CHC+PT=فَعِلَ+CI=ia+FLX=iaCHC+DRV=DCHC:فَعَل;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiDCC
رَؤُفَ,V+TYP=Tri+RT=رأف+TR=Intr+FRM=VN0+RI=CHC+PT=فَعُلَ+CI=uu+FLX=uuCHC+DRV=DCHC:فَعُلَة;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiD
اِرْتَالَ,V+TYP=Qua+RT=رأل+TR=Intr+FRM=VN0+RI=CHC+PT=اِفْتَعَلَ++DRV=DCHC:اِفْتِعَال;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiDCCC:مُفْتَعِل;FLXDRV+DRV=SDCCC:يَة;FLXDRV
رَاءَلَ,V+TYP=Qua+RT=رأل+TR=Intr+FRM=VN0+RI=CHC+PT=فَاعَلَ++DRV=DCHC:مُفَاعَلَة;FLXDRV+DRV=DCHC:فَاعَلَ;MaFLXواحدة+DRV=MiDCCC:مُفَاعِل;FLXDRV+DRV=SDCCC:يَة;FLXDR
اِسْتَرْأَلَ,V+TYP=Qua+RT=رأل+TR=Tr+FRM=VN0N1+RI=HCC+PT=اِسْتَفْعَلَ++DRV=DHCC:اِسْتِفْعَال;FLXDRV+DRV=DHCC:اِسْتَفْعَلَ;MaFLXة+DRV=MiDHCC:مُسْتَفْعِل;FLXDRV+DRV=SDCCC:يَة;FLX
رَأْمَ,V+TYP=Tri+RT=رأم+TR=Tr+FRM=VN0N1+RI=CHC+PT=فَعَلَ+CI=aa+FLX=aaCHC+DRV=DCHC:فَعَل;FLXDRV+DRV=HDCHC:فَعْلَة;FLXDRV+DRV=MaDCCC:فَعْلَة;FLXDRV+DRV=MiDC
رَئِمَ,V+TYP=Tri+RT=رأم+TR=Tr+FRM=VN0N1+RI=CHC+PT=فَعِلَ+CI=ia+FLX=iaCHC+DRV=DCHC:فَعَلان;FLXDRV+DRV=DCHC:فَعَلان;FLXDRV+DRV=DCHC:فَعَل;FLXDRV+DRV=HDCHC
اِرْأَمَّ,V+TYP=Qua+RT=رأم+TR=Tr+FRM=VN0N1+RI=CHC+PT=اِفْعَلَّ++DRV=DCHC:اِفْعَال;FLXDRV+DRV=DCHC:اِفْعَلَّ;MaFLXة+DRV=MiDCCC:مُفْعَلّ;FLXDRV+DRV=SDCCC:يَة;FLXDRV
```

Figure 5 The constructed dictionary

As stated before, we saved the names of the derivational and inflectional graphs in our Excel File which allows us to use Excel's string and logical functions like concatenate() and conditional if() to build quick and easy lines to use in our dictionary. This allows us to achieve this constructed Nooj dictionary as efficiently and less time consuming as possible. The Figure 6 shows a preview of the structure used which would help us later.

FLX	DRV	Nooj	MSD DRV	HMSD DRV	MaMSD DRV	MiMSD DRV	SMSD DRV
FLX=aaCHC	DRV=DCHC:فَعْل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فَعْل	DRV=HDCHC:فَعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة
FLX=aaCHC	DRV=DCHC:فَعْل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فَعْل	DRV=HDCHC:فَعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة
FLX=uuCHC	DRV=DCHC:فُعُولَة	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فُعُولَة	DRV=HDCHC:فَعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة
FLX=اِفْتَعَلCHC	DRV=DCHC:اِفْتِعَال	FLXDRV:يَةFLXDRV+DRV=SDCCC:مُفْتَعَل	DRV=DCHC:اِفْتِعَال				DRV=SDCCC:يَة
FLX=تَفَاعَلCHC	DRV=DCHC:تَفَاعُل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مُتَفَاعَل	DRV=DCHC:تَفَاعُل		MaFLXة	DRV=MiDCCC:مُتَفَاعَل	DRV=SDCCC:يَة
FLX=تَفَعَّلCHC	DRV=DCHC:تَفْعِيل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مُتَفَعَّل	DRV=DCHC:تَفْعِيل		MaFLXة	DRV=MiDCCC:مُتَفَعَّل	DRV=SDCCC:يَة
FLX=فَعْلَلHDD	DRV=DHDD:فَعْلَلَة	FLXDRV:يَةFLXDRV+DRV=SDHDD:فَعْلَل	DRV=DHDD:فَعْلَلَة		MaFLXة	DRV=MiDCCC:مُفَعْلَل	DRV=SDHDD:يَة
FLX=aaCHC	DRV=DCHC:فِعَالَة;FLXDRV+	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فِعَالَة+DRV	DRV=HDCHC:فِعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة
FLX=iaCHC	DRV=DCHC:فَعَل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فَعَل	DRV=HDCHC:فِعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة
FLX=فَاعَلCHC	DRV=DCHC:مُفَاعَلَة	FLXDRV:يَةFLXDRV+DRV=SDCCC:مُفَاعَل	DRV=DCHC:مُفَاعَلَة		واحدةMaFLX	DRV=MiDCCC:مُفَاعَل	DRV=SDCCC:يَة
FLX=اِفْتَعَلCHC	DRV=DCHC:اِفْتِعَال	FLXDRV:يَةFLXDRV+DRV=SDCCC:مُفْتَعَل	DRV=DCHC:اِفْتِعَال				DRV=SDCCC:يَة
FLX=تَفَعَّلCHC	DRV=DCHC:تَفْعِيل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مُتَفَعَّل	DRV=DCHC:تَفْعِيل		MaFLXة	DRV=MiDCCC:مُتَفَعَّل	DRV=SDCCC:يَة
FLX=aaCHC	DRV=DCHC:فَعْل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فَعْل	DRV=HDCHC:فَعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة
FLX=iaCHC	DRV=DCHC:فَعَل	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فَعَل	DRV=HDCHC:فَعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة
FLX=uuCHC	DRV=DCHC:فَعَالَة	FLXDRV:يَةFLXDRV+DRV=SDCCC:مَفْعَل	DRV=DCHC:فَعَالَة	DRV=HDCHC:فَعْلَة	DRV=MaDCCC:فَعْلَة	DRV=MiDCCC:مَفْعَل	DRV=SDCCC:يَة

Figure 6 Dictionary constructed from Excel file

- **Nooj:** Here represents the lines of the dictionary.
- **FLX:** This represents the flectional graph of the verb.
- **MSD DRV:** This represents the derivational graph for the Original gerund (Mojared /Assli-مجرّد\أصلي) corresponding to the verb.
- **HMSD DRV:** This represents the derivational graph for the State gerund (Hayea-الهيئة) corresponding to the verb.
- **MaMSD DRV:** This represents the derivational graph for the One-Time gerund (Almarra -المرّة) corresponding to the verb.
- **MiMSD DRV:** This represents the derivational graph for the M-gerund (Mimi-الميمي) corresponding to the verb.
- **SMSD DRV:** This represents the derivational graph for the Industrial gerund (Alsinaa'i-الصناعي) corresponding to the verb.

By using the aforementioned Excel functions, we can concatenate these strings to construct the dictionary line by line. This way only a temporary method due to time constraints and we could use another process that is faster and more error proof.

Conclusion: We now have a lexical dictionary that has for each entry: The verb, its root, type, pattern, root info, conjugation info, and form. In addition, we also include the inflectional and derivational graphs for each entry.

2.2.2 Inflectional & Derivational Grammars

In order to analyze texts, NooJ needs dictionaries which house and describe all of the words of that text, as well as some mechanism to link these lexical entries to all the corresponding (inflected and/or derived) forms that actually occur in text. NooJ offers two equivalent tools to represent and describe inflections and derivations:

- **Inflectional / derivational descriptions** are organized sets of Context- Free rules that describe morphological paradigms.
- **Inflectional / Derivational grammars** are structured sets of graphs that describe morphological paradigms.

Both sets of rules are stored in NooJ inflectional/derivational grammars, in ".nof" files. These descriptions are lexicalized, i.e. each lexical entry of a NooJ dictionary must be associated with one or more inflectional and derivational paradigms, and inflectional or derivational paradigms do not work if they are not associated with lexical entries.

a- Inflectional Grammar

NooJ's inflection module is triggered by adding the special property **"+FLX**" to a lexical entry. For instance, in the our dictionary, we see the following entries:

رأب,رَأَبَ, V+Tr+VN0N1+CHC+فَعَلَ+aa+FLX= aaCHC
رأد,رَأَدَ, V+Intr+VN0+CHC+فَعَلَ+aa+FLX= aaCHC
رأد,رَؤُدَ, V+Intr+VN0+CHC+فَعُلَ+uu+FLX= uuCHC

This sample of our dictionary states that lexical entries رَأَبَ, رَأَدَ share the same inflectional graph, named "**aaCHC**", while the lexical entry رَؤُدَ is associated with the graph named "**uuCHC**".

NooJ provides two equivalent tools to describe these inflectional paradigms: either graphically or by means of (textual) rules. We use the graphical rules to clarify and facilitate the understanding of these rules thanks to NooJ's graphical editor.

The FLX paradigm represents all inflectional forms (inflection forms for verbs or gerunds) for each dictionary entry. These FLXs are represented using our defined rules graphs as Figure 7 shows. We preferred to describe this inflectional paradigms using NooJ's graphical rules interface that is equivalent to the textual rule editor, here is our inflectional paradigms that are assigned to the dictionary. Each dictionary entry has inflectional paradigms and derivational paradigms, that generate all its inflectional and derivational forms.

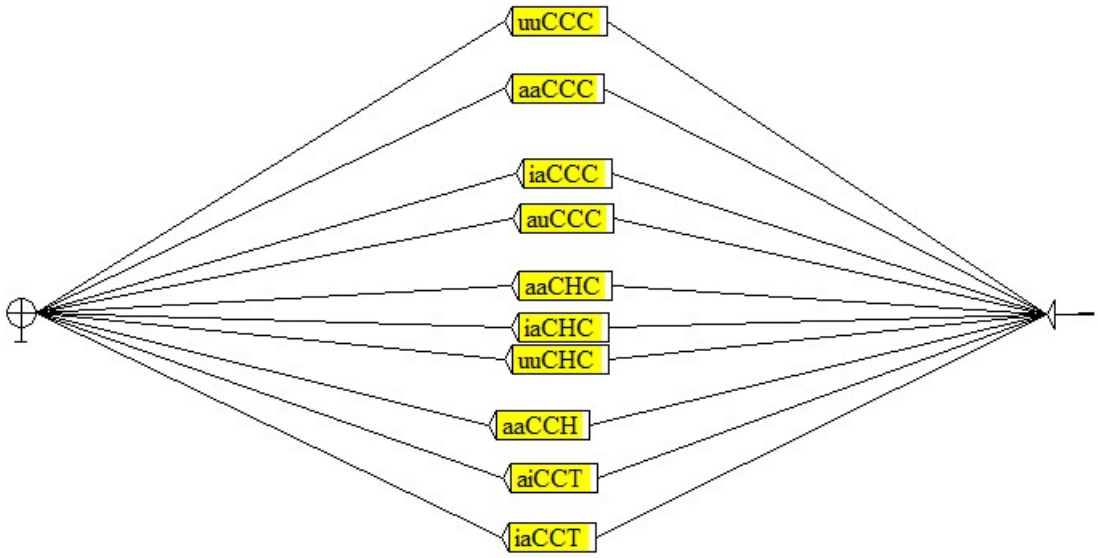

Figure 7 Inflectional Paradigm for Subjunctive Case

For Example the lexical entry (to nurture – رَأَبَ) has an inflectional paradigm: **"aaCHC**" that matches any form in the set of {you nurture - تَرْأَبُوا – TaR'aBou, he nurture - يَرْأَبَ – YaR'aB,..}

NooJ recognizes all this forms even if they are semi or non- vowelized. Each verb is conjugated in the subjunctive case or المضارع المنصوب - present tense (Mansoub) as it is shown in Figure 7. We used the pre-defined NOOJ operators to define the inflections of these entries. There are also some special operators <Z>,<T> AND <M> that are defined only for Arabic language for more information please read NooJ manual.

Figure 8 shows how we easily define the present tense (Mansoub)-المضارع المنصوب for the verb (to nurture – رَأَبَ) using NooJ's predefined operators. For instance <B> : will erase the last character, <Z> will add ت character while the last character of the given verb is not a ت, else it will add تُ ...etc. As it's mentioned above this graph recognizes all appearances of the present conjugation set { I nurture , you nurture ,...}, and annotate them with the morphological defined properties which appear under each node for instance { S+1+s } : the first singular person in active voice . S: Subjunctive / 1: first person and s : singular .

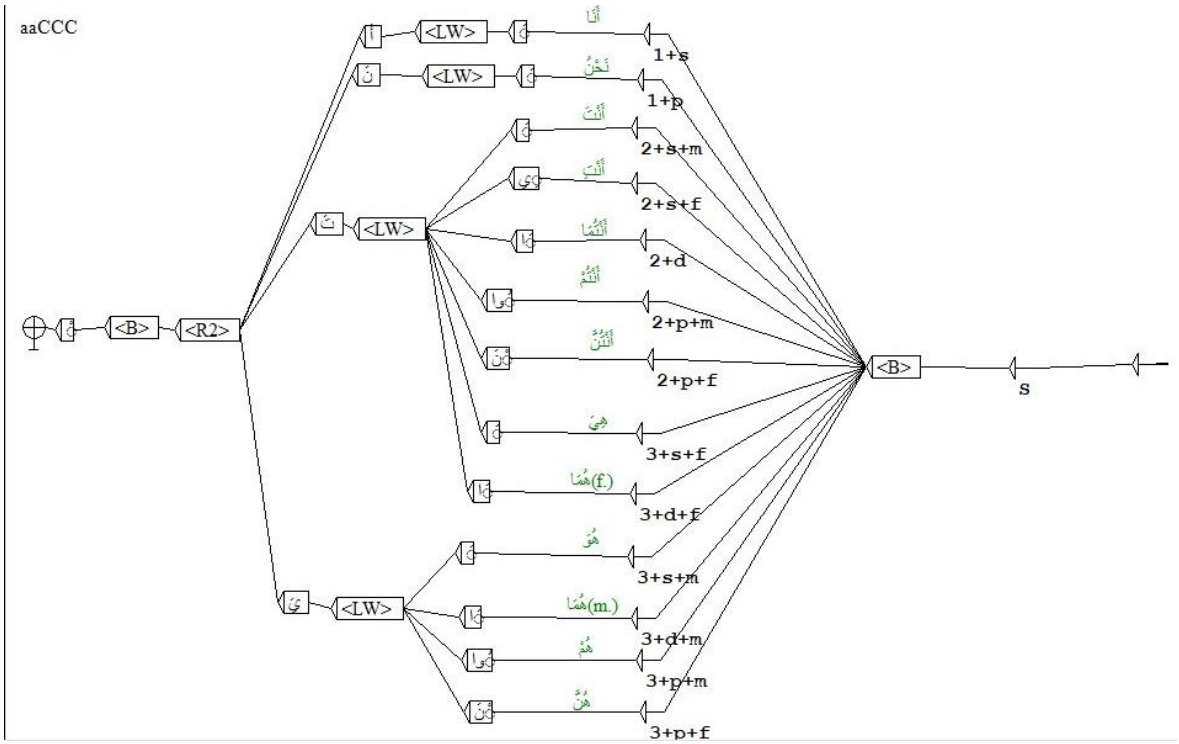

Figure 8 Inflectional Graph - "aaCCC"

b- Derivational Grammar

Inflectional grammars (both graphical and textual) can include derivational paradigms. Derivational paradigms are very similar to inflectional paradigms, except that the morpho-syntactic category of each word form must be explicitly produced by derivational transducers. The special information "**+DRV**" is used to indicate a derivational paradigm.

Let us consider the following lexical entry from our dictionary:

رأب,رَأَبَ,V+Tr+VN0N1+CHC+فَعَلَ+aa+DRV=DCHCفَعْل:FLXDRV

This entry makes NooJ produce the original gerund (Mojared/Assli) "رَأْب" according to the derivational paradigm "**DCHCفَعْل**", which is then inflected according to the paradigm "**FLXDRV**". The forms "رَأْبَ", "رَأْبُ","رَأْبِ ","رَأْبً ","رِأْبٌ","رِأْبٍ " will then be lemmatize as "رأب", even though they will be associated with the category "MSD". In consequence, query such as <رَأَبَ> will match both the conjugated forms of the verb "رَأَبَ", as well as its derived forms, including the gerund "رَأْب" and its derived forms.

An expected case in our dictionary is:

رأب,رَأَبَ,V+Tr+VN0N1+CHC+فَعَلَ+aa+FLX=V_CHCaa+DRV=DCHCفَعْل:FLXDRV

This states that the verb "رَأَبَ" inflects according to the inflectional paradigm "**V_CHCaa**", and then derives according to the derivational paradigm "**DCHCفَعْل**". The latter paradigm produces the original gerund (Mujared/Assli) "رَأْب", which then inflects according to the default inflectional paradigm "**FLXDRV**".

The DRV paradigm represents all derivational forms producing the different types of gerunds for each dictionary entry. These DRVs are also represented using our defined rules graphs as Figure 9 shows. Here is our derivational paradigms that are assigned to the dictionary.

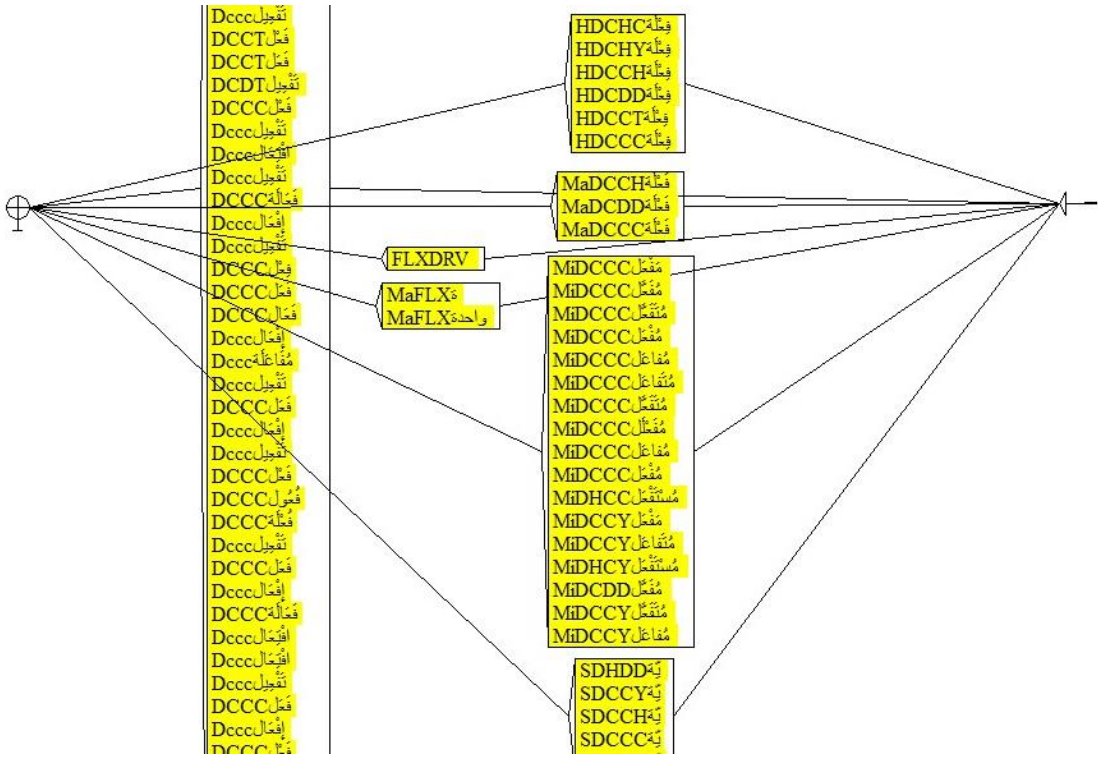

Figure 9 Derivational Paradigm

For Example the lexical entry (to nurture – رَأَبَ) has different derivational paradigms for each different gerund type that matches any form in the set of:

- **"DRV=DCHCفَعْل":** This produces the **original gerund** (Mojared/Assli-مجرّد\أصلي). [رَأْب,MSD]
- **"DRV=HDCHCفِعْلَة":** This produces the **state gerund** (Hayea-الهيئة). [رأْبَة,HMSD]
- **"DRV=MiDCCCمَفْعَل":** This produces the **M-gerund** (Mimi-الميمي). [مَرْأَب,MIMSD]
- **"DRV=MaDCCCمَفْعَلَة":** This produces the **one-time gerund** (Almarra-المرّة). [رَأْبَة,MAMSD]
- **"DRV=SDCCCيّة":** This produces the Industrial gerund (Alsinaa'i-الصناعي). [رَأْبِيّة,SMSD]

NooJ recognizes all these forms even if they are semi or non- vowelized. Each verb is converted into the respective gerunds as it is shown in Figure 9. We also use the pre-defined NOOJ operators and the special operators that are defined only for Arabic language to define the derivations of these entries. Figure 10 shows how we define a general inflectional graph "**FLXDRV**" for the most of the gerunds except for the [MAMSD] **one-time gerund** (Almarra-المرّة) which can be of two types, the one we need to add a ("t"/ة) to, or the one we need to add ("once"/واحدة) to. Only after that, it could apply the general inflections by passing through the path of the general flectional graph"**FLXDRV**".

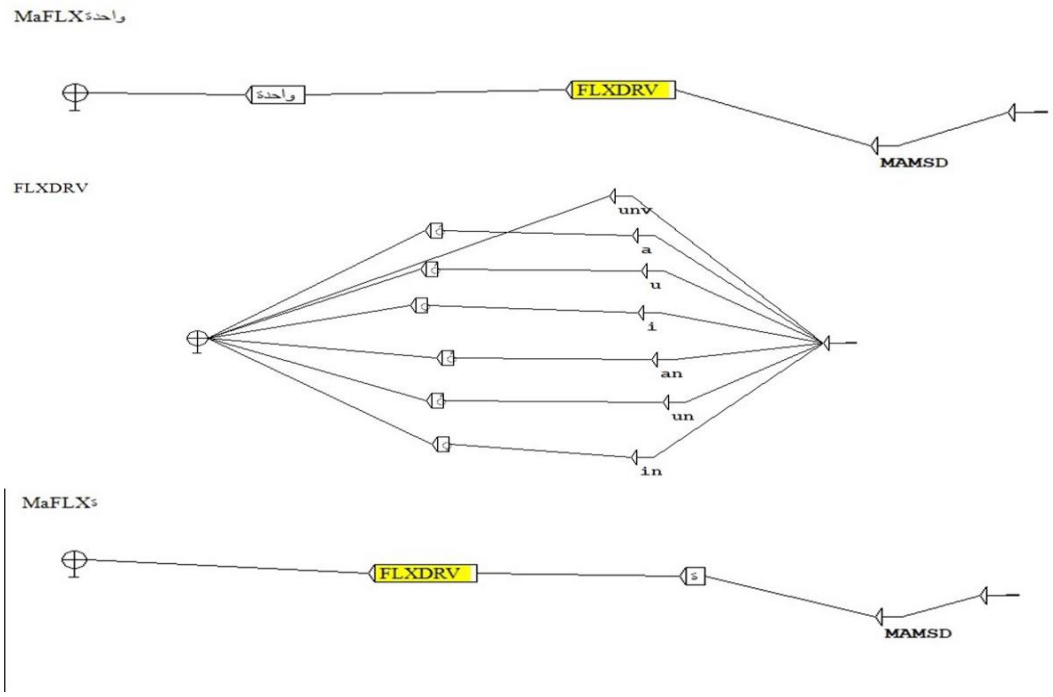

Figure 10 Inflectional Paradigm for Gerunds

Figure 11 shows how we define the corresponding gerund for the verb (to nurture – رَأَبَ) using NooJ's predefined operators and annotate them with the morphological defined properties which appear under each node which come in these category forms {MSD, HMSD, SMSD, MAMSD, MIMSD}.

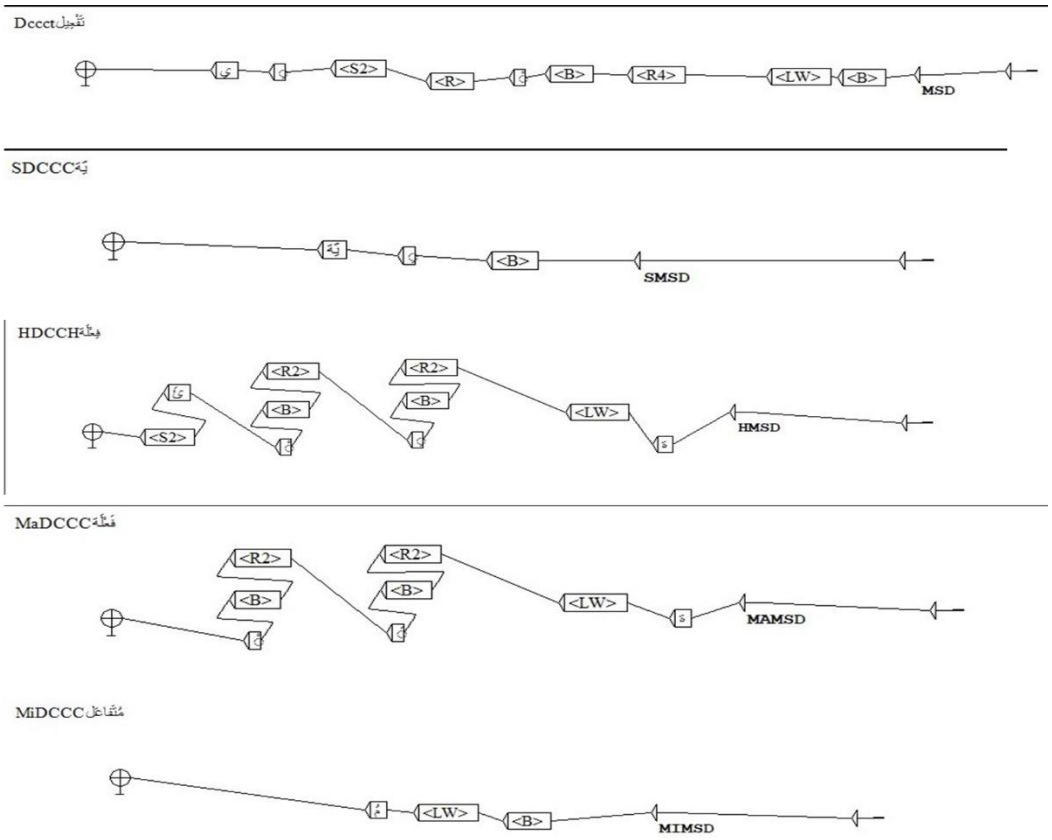

Figure 11 Derivational Graphs for Types of Gerund

- **MSD**= original gerund; (Mojared /Assli-مجرّد\أصلي)
- **HMSD**= state gerund; (Hayea-الهيئة)
- **SMSD**= industrial gerund; (Alsinaa'i-الصناعي)
- **MASD**= one-time gerund; (Almarra -المرّة)
- **MIMSD**= mimi gerund; (Mimi-الميمي)

Conclusion: We now have two grammars. The Inflectional grammar "V_grammar", that conjugates a given verb from the entry of our dictionary to the subjunctive case, and the derivational grammar "D_grammar" that derives the verbs to the different types of gerunds. Compiling the dictionary gives us 2605 word forms.

2.3 Syntactical Analysis using NooJ

We saw that certain parts of the dictionary could be processed with inflectional and derivational grammars. These grammars were applied by NooJ's morphological engine to simple word forms (sequences of letters between delimiters); hence, they could not process multi-word units or frozen expressions. The syntactic grammars we see now are also used to represent ALUs (Atomic Linguistic Units), but these ALUs can be multi-word units or frozen expressions.

2.3.1 Syntactic Grammar

We want to identify some specific determiners written out in text form (e.g. verb, gerund…). Thus, we create a new syntactic grammar in NooJ and name it "**A_grammar**", which allows us to specify how the specific determiners we want should be handled and annotated. The main graph of the syntactic grammar, includes links to three embedded graphs: "**VMSD**", "**MSDV**" and "**ANV**", as seen in the following Figure 12:

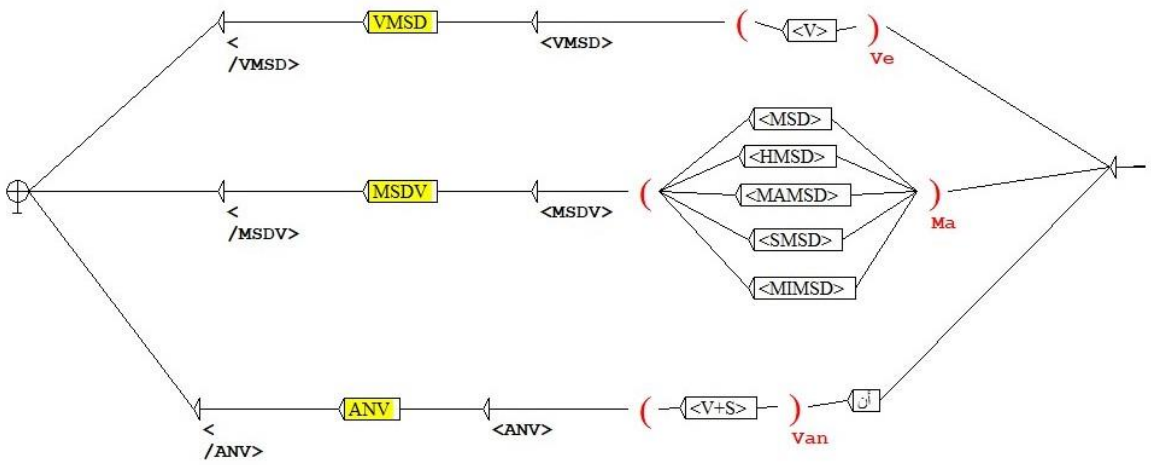

Figure 12 Syntactical Paradigm

As shown in the previous Figure 12, we have three different paths in the main graph. These paths dictate how to handle the discovered determiners with are in this case:

- <V> for verbs;
- <MSD>, <HMSD>, <MAMSD>, <MIMSD>, <SMSD> are for the gerunds (massadir);
- <V+S> for verbs in the subjunctive case (preceded by " أن ");

We enhance this grammar so that instead of merely recognizing these determiners, NooJ will tag them in texts, just as if they had been entered in a dictionary. We want to annotate the matching sequences depending on the path taken. In order to do that, we give the paths taken their respective identifiers: **<VMSD>, <MSDV>, <ANV>.** The following Figures 13, 14, and 15 give an overview of the structure used for these annotations.

- **First Path (VMSD):** Identified by **<VMSD> </VMSD>,** which takes a verb **<V>** as an entry and stores it in a variable "**Ve**". Then the sub-graph "**VMSD**" handles the inputted verb by annotating it using pre-defined tags provided from us and extracts the desired information, which this case we extract the different gerunds from the verb that is stored in the variable "**Ve**". [Figure. 13]

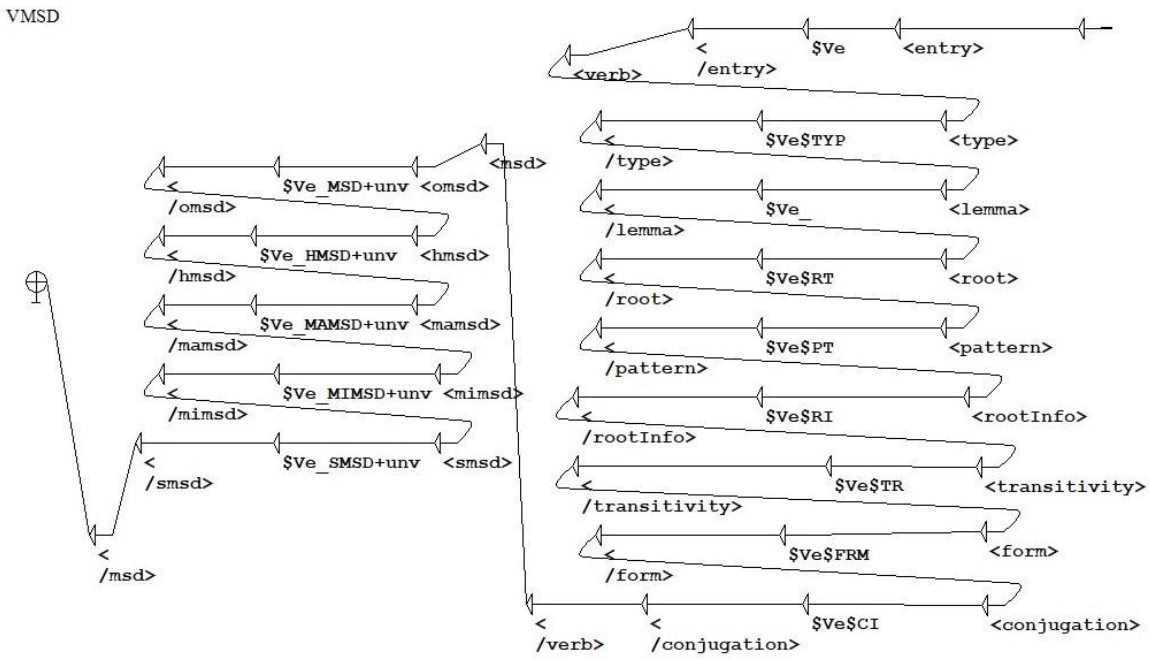

Figure 13 Syntactical Paradigm of the " VMSD" Sub-Graph

- **Second Path (MSDV):** Identified by **<MSDV> </MSDV>,** which takes any type of gerund **<MSD>, <MSD>, <HMSD>, <MAMSD>, <MIMSD>, or <SMSD>** as an entry and stores it in a variable "**Ma**". Then the sub-graph "**MSDV**" handles the inputted gerund by annotating it using pre-defined tags provided from us and extracts the desired information, which this case we extract the original verb from the gerund that is stored in the variable "**Ma**". [Figure. 14]

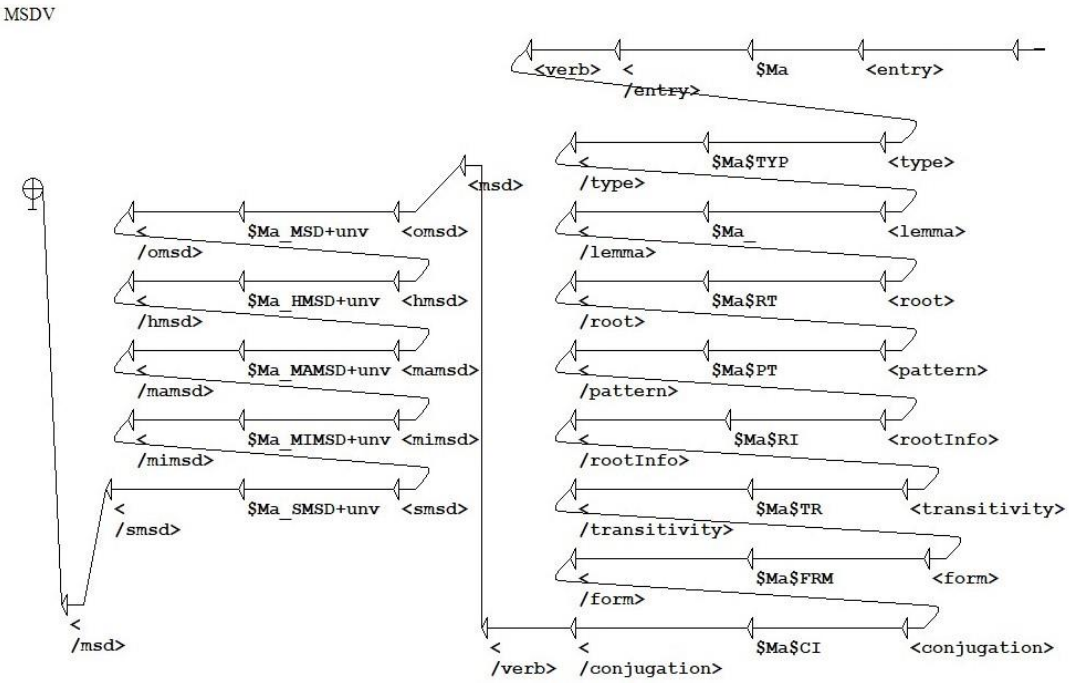

Figure 14 Syntactical Paradigm of the " MSDV" Sub-Graph

- **Third Path (ANV):** identified by **<ANV> </ANV>,** which takes a verb in the subjunctive case **<V+S>** preceded by " أن " as an entry and stores it in a variable "**Van**". Then the sub-graph "**ANV**" handles the inputted verb by annotating it using pre-defined tags provided from us and extracts the desired information, which this case we extract the appropriate gerund from the verb that is stored in the variable "**Van**". [Figure. 15]

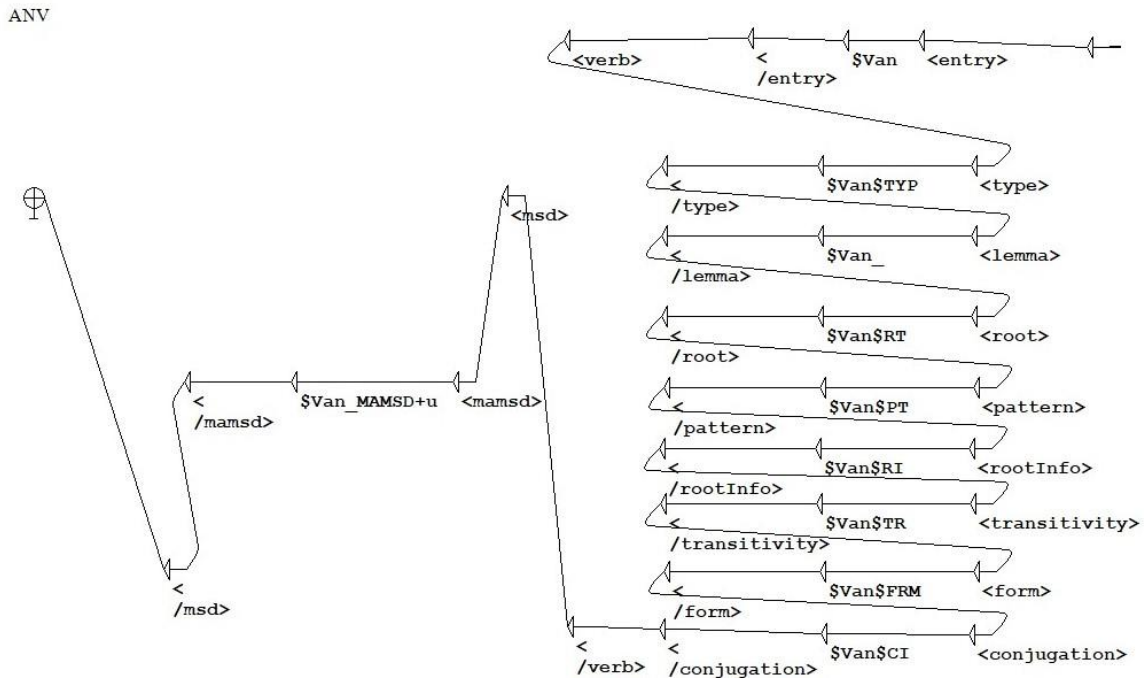

Figure 15 Syntactical Paradigm of the " ANV" Sub-Graph

As the previous figures (13, 14, 15) show, we chose to add these annotations in a way that transforms the result from these paths to a bloc of XML that could be easily used by our application. We discuss more about this in the upcoming sections.

2.4 Noojapply

Once our dictionary is compiled, we can use it as resource to analyze any Arabic text that contains verbs and gerunds or their inflectional forms. We then resort to use Noojapply. Noojapply is a command-line program, which can be called either directly from a "shell" script, or from more sophisticated programs written in PERL, C++, JAVA, etc. In our work, we use JAVA. It allows us to apply the given lexical resources (Dictionary with inflectional and derivational grammars) and then the syntactic resources (Syntactic grammar) to each text, annotates them, and then produce the .ind text file, which contains the desired results in an XML format. As we see in Fig. 16, Noojapply uses the provided resource to analyze a given text; it also returns the annotations, which contains the linguistic details about the given text. Noojapply also returns the annotations as an XML structure. We considered this file as the main input in our Java application. Many ideas could be achieved using the XML annotations, because it tells you about the position, linguistic features and the number of any verbs, gerunds, or other words occurred in the text, in other word the annotation file reflects how you represent Arabic words in your resource.

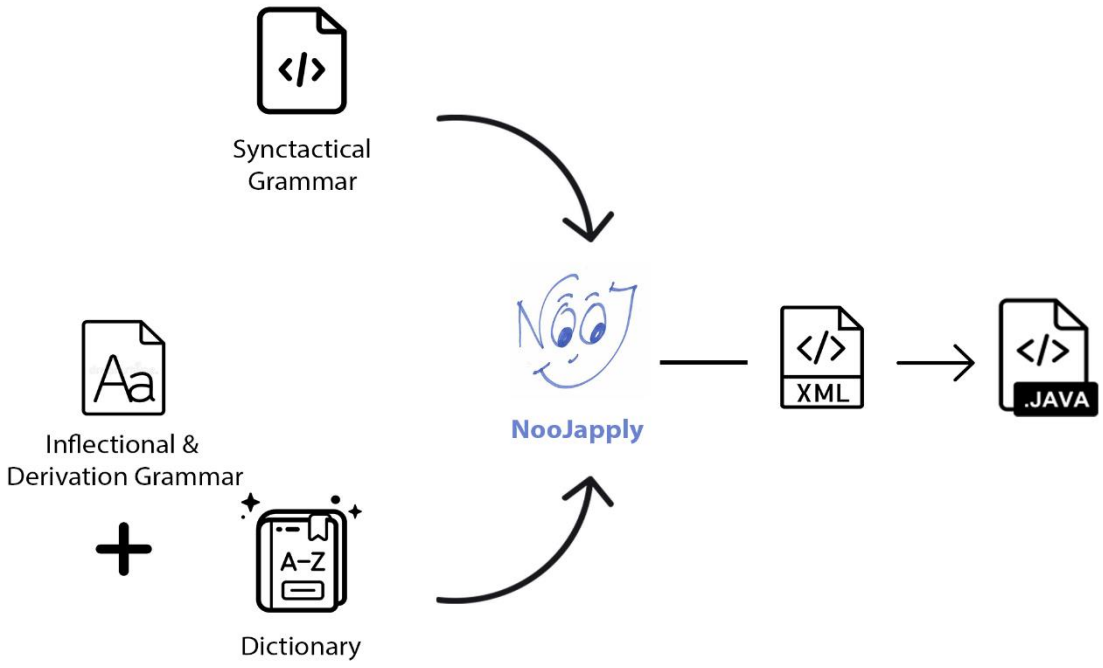

Figure 16 Noojapply & Arabic Resources

As previously mentioned, we defined our own XML tags in the syntactical grammar in order to use them in our Java application. As a result, Noojapply returns annotations as an XML structure to standardize the output and make it readable and compatible with our programming language (Java).

For instance, we run the following Noojapply command in the CMD:

```
noojapply ar result.ind bigdic.nod A_grammar.sft textToAnalyze.txt
```

We also provide the following files in the same folder:

- **BigDic.nod:** This is our dictionary. It contains our verb + their linguistic information.
- **V_grammar:** This is our inflectional grammar. It contains our inflectional graphs.
- **D_grammar:** This is our derivational grammar. It contains our derivational graphs.
- **A_grammar:** This is our Syntactical grammar. It contains our annotation graphs.
- **Text.txt:** This contains the text that we want to analyse.
- **Result.ind:** This is our output/results file (It is generated automatically).

Figure 17 shows the given text file to analyse.

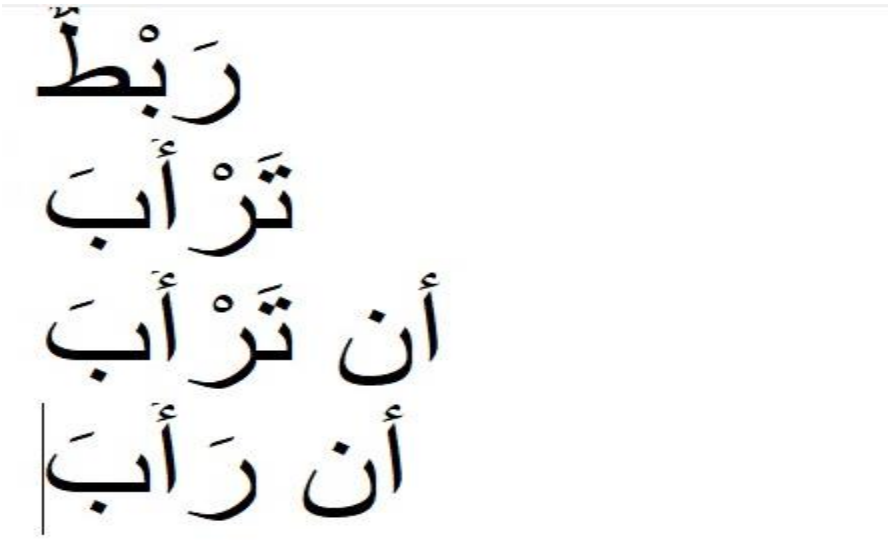

Figure 17 Example of Arabic Text to Analyze

Figure 18 shows the structure of the results file.

```
0,6,<MSDV  >
    <entry>بَيْطٌ</entry  >
    <verb    >
        <type>Tri</type>
        <lemma>بَيْطَ</lemma>
        <root>بيط</root    >
        <pattern>فَعَلَ</pattern>
        <rootInfo>CCC</rootInfo    >
        <transitivity>Tr</transitivity>
        <form>VN0N1</form>
        <conjugation>au</conjugation>
    </verb  >
    <msd    >
        <omsd>بَيْط</omsd>
        <hmsd>بَيْطَة</hmsd>
        <mamsd>بَيْطَةَ</mamsd    >
        <mimsd>مَبْيَط</mimsd>
        <smsd>بَيْطَرِيَّة</smsd>
    </msd  >
</MSDV
>
0,8,<VMSD  >
<entry>رَأَبَ/entry  >
<verb    >
    <type>Tri</type>
    <lemma>رَأَبَ</lemma>
    <root>رأب</root    >
    <pattern>فَعَلَ/pattern>
    <rootInfo>CHC</rootInfo    >
    <transitivity>Tr</transitivity>
    <form>VN0N1</form>
    <conjugation>aa</conjugation>
</verb  >
<msd    >
    <omsd>رَأْب</omsd>
    <hmsd>رَيْأَة</hmsd>
    <mamsd>رَيْأَةَ</mamsd    >
    <mimsd>مَرْأَب</mimsd>
    <smsd>رَيْئِيَّة</smsd>
</msd  >
</VMSD
>
0,11,<ANV  >
<entry>رَأَبَ/entry  >
<verb    >
<type>Tri</type>
<lemma>رَأَبَ</lemma>
<root>رأب</root    >
<pattern>فَعَلَ/pattern>
<rootInfo>CHC</rootInfo    >
<transitivity>Tr</transitivity>
<form>VN0N1</form>
<conjugation>aa</conjugation>
</verb  >
<msd>
<mamsd>رَيْأَةُ</mamsd>
</msd>
</ANV
>
*****
```

Figure 18 Noojapply Results File (.ind)

We can distinguish three main categories / tags:

- **<VMSD> </VMSD>:** This represents the “Verb to Gerund” conversion.
- **<MSDV> </MSDV>:** This represents the “Gerund to Verb” conversion.
- **<ANV> </ANV>:** This represents the “ان + Verb in Subjunctive] to Gerund” conversion.

As stated before, describe the taken path in the syntactical grammar and the conversion type for the application. Inside these main tags, we find three more tags that are distinct:

- **<entry> </entry>:** This contains the entry. It may be either a verb or gerund.
- **<verb> </verb>:** This contains the verb and its linguistic info:
    - **<type> </type>:** This field contains the type of the verb (Tri / Qua).
    - **<lemma> </lemma>:** This field contains the original verb.
    - **<root> </root>:** This field contains the root of the verb.
    - **<pattern> </pattern>:** This field contains the pattern of the verb.
    - **<rootInfo> </rootInfo>:** This field contains a description of the root.
    - **<transitivity> </transitivity>:** This describes the transitivity of the verb.
    - **<form> </form>:** This describes the phrase structure of the verb.
    - **<conjugation> </conjugation>:** This describes the conjugation of the verb.
- **<msd> </msd>:** This contains all the types of gerunds corresponding to the verb:
    - **<omsd> </omsd>:** This contains the original gerund (Mojared /Assli-أصلي\مجرّد).
    - **<mamsd> </mamsd>:** This contains the one-time gerund (AImarra -المرّة).
    - **<hmsd> </hmsd>:** This contains the industrial gerund (AIsinaa’i-الصناعي).

- **<mimsd> </mimsd>:** This contains the M-gerund (Mimi-الميمي).

In this structure, the blocs specify the used conversion paradigm for each analyzed word; it also gives an object that contains both the verb and its linguistic feature in addition to the different types of corresponding gerunds. This file is considered as the input for our Java application.

Conclusion: Now that we have our compiled dictionary, the inflectional grammar, the derivational grammar, and the syntactical grammar, we are able to analyze any given Arabic text, extract the desired information as an XML file, which is then used by our application to format, and display this information to the user.

Chapter 3: **JAVA Application**

We set out to make a Java Application that would be able to analyse any Arabic text and extract whatever the user chooses (verb, gerunds or noun phrases). Our application starts with a few main ideas and features like being able to read a text and extract the verbs and gerunds in it with the help of Noojapply and our linguistic resources. On top of that we should make an application that lives up to our current standards in terms of combining a sleek UI (User Interface) and a smooth UX (User Experience). In addition to achieving the main features, we were able to enhance the accessibility and usability of our application thanks to small QoL (Quality of Life) changes. We are going to follow the steps taken to create our Java application from conception to realization and implementation in the following sections.

3.1 Application Conception & Modelling

As a first benchmark, we set the features that our application should have. Then we start with an overview of the application to determine how the UI and UX are going to play along with each other. Followed by that, we use UML (Unified Modeling Language), which is a general-purpose, developmental, modeling language in the field of software engineering, to provide a standard way to visualize the design of our application system.

3.1.1 Application Features

This gives us an idea on the feature we want our application to have and then use them as programming goals to measure our progress. The Java application supports the following features:

- Analyse an Inputted text on a text field and display the gerunds and verbs in it.
- Analyse a text (.txt), PDF (.pdf) or document (.docx) file and display the gerunds and verbs in it.

- Extract the different types of gerund from a given verb.
- Extract the original verb from any given type of gerund.
- Convert an Arabic verb phrase to a noun phrase.
- Display the linguistics information of a given verb.
- Allow for multiple tabs for multi-tasking.
- Search and filter our results.

3.1.2 Unified Modelling Language (UML)

UML, short for Unified Modelling Language, is a standardized modelling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems. The UML is a very important part of developing object oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

According to the Object Modelling Group (OMG), "modelling is **the designing of software applications before coding**." In model-based software design and development, software modelling is used as an essential part of the software development process. In the next section, we use UML to describe the different structure and concepts of our Java application before programming it.

3.1.3 Modelling Tool

Visual Paradigm (VP-UML) is a UML CASE Tool supporting UML 2, SysML and Business Process Modelling Notation (BPMN) from the Object Management Group (OMG). In addition to the modelling support, it provides report generation and code engineering capabilities including code generation. It can reverse engineer diagrams from code, and provide round-trip engineering for various programming languages.

3.1.4 Use Case Diagram

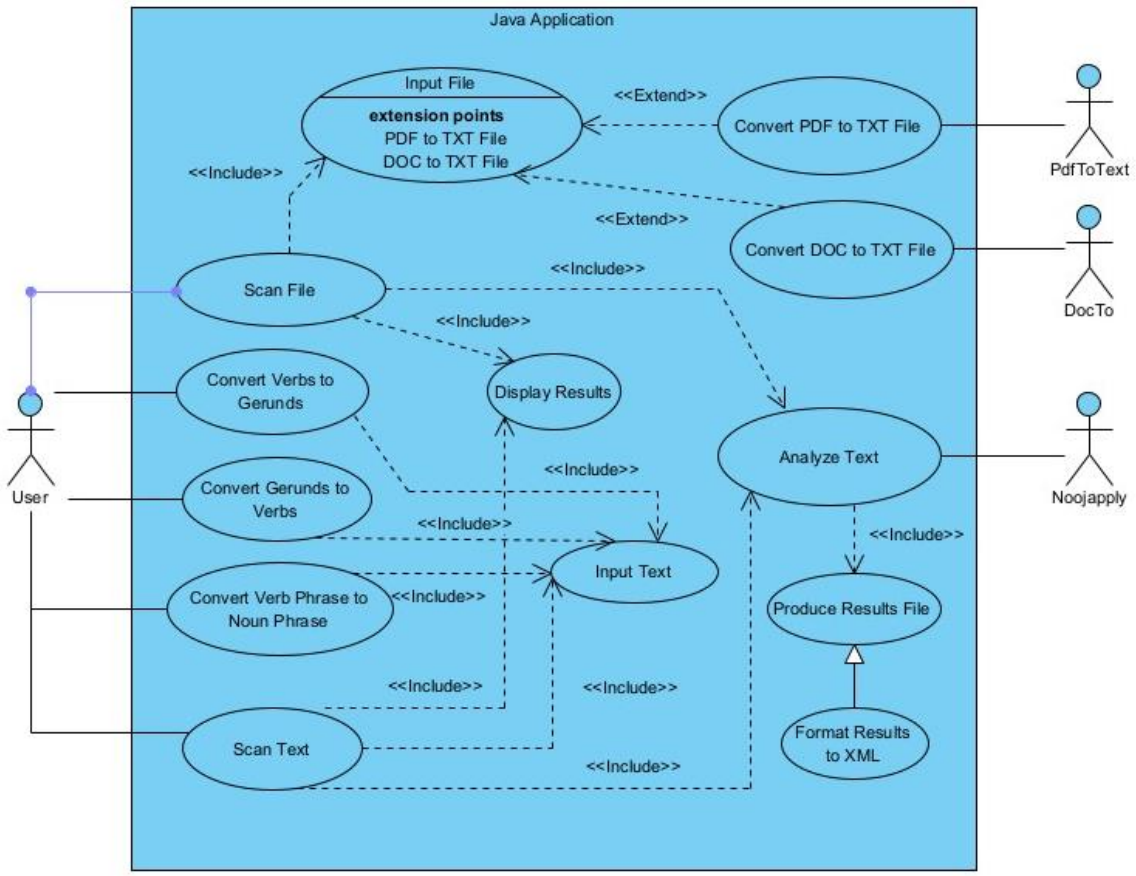

Figure 19 Use Case Diagram

3.1.5 Class Diagrams

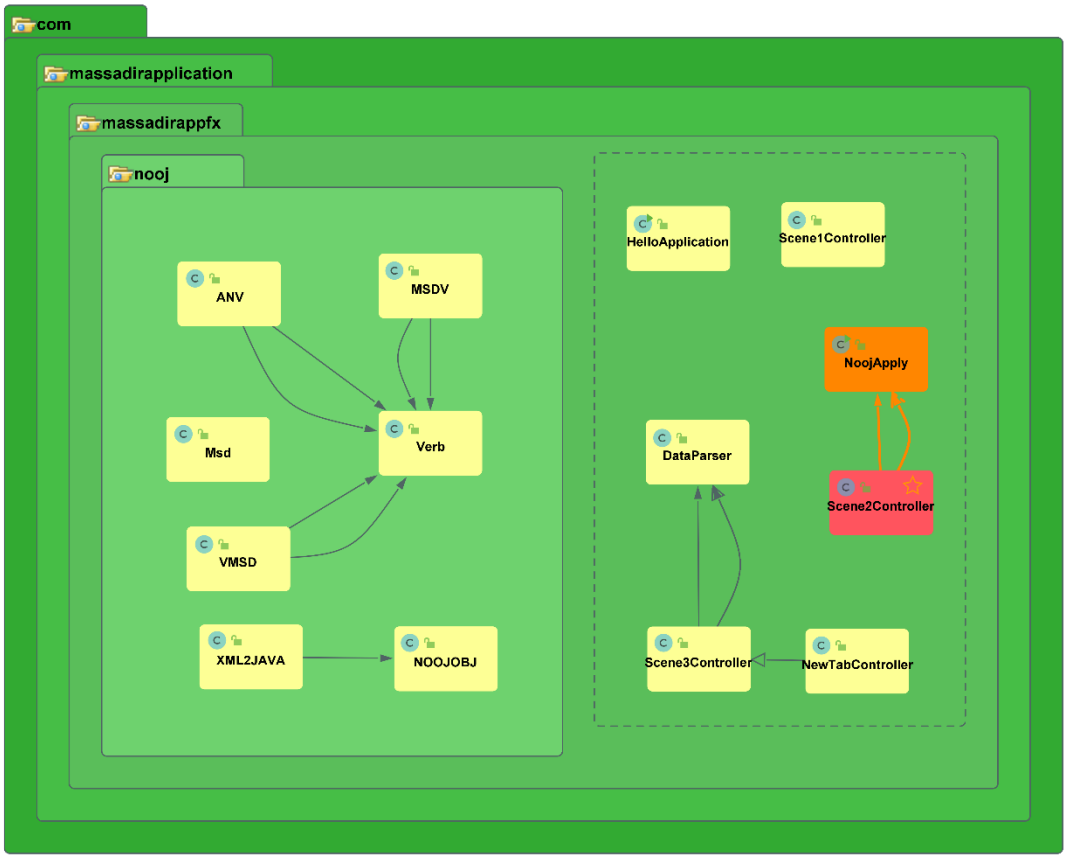

Figure 20 General Class Diagram

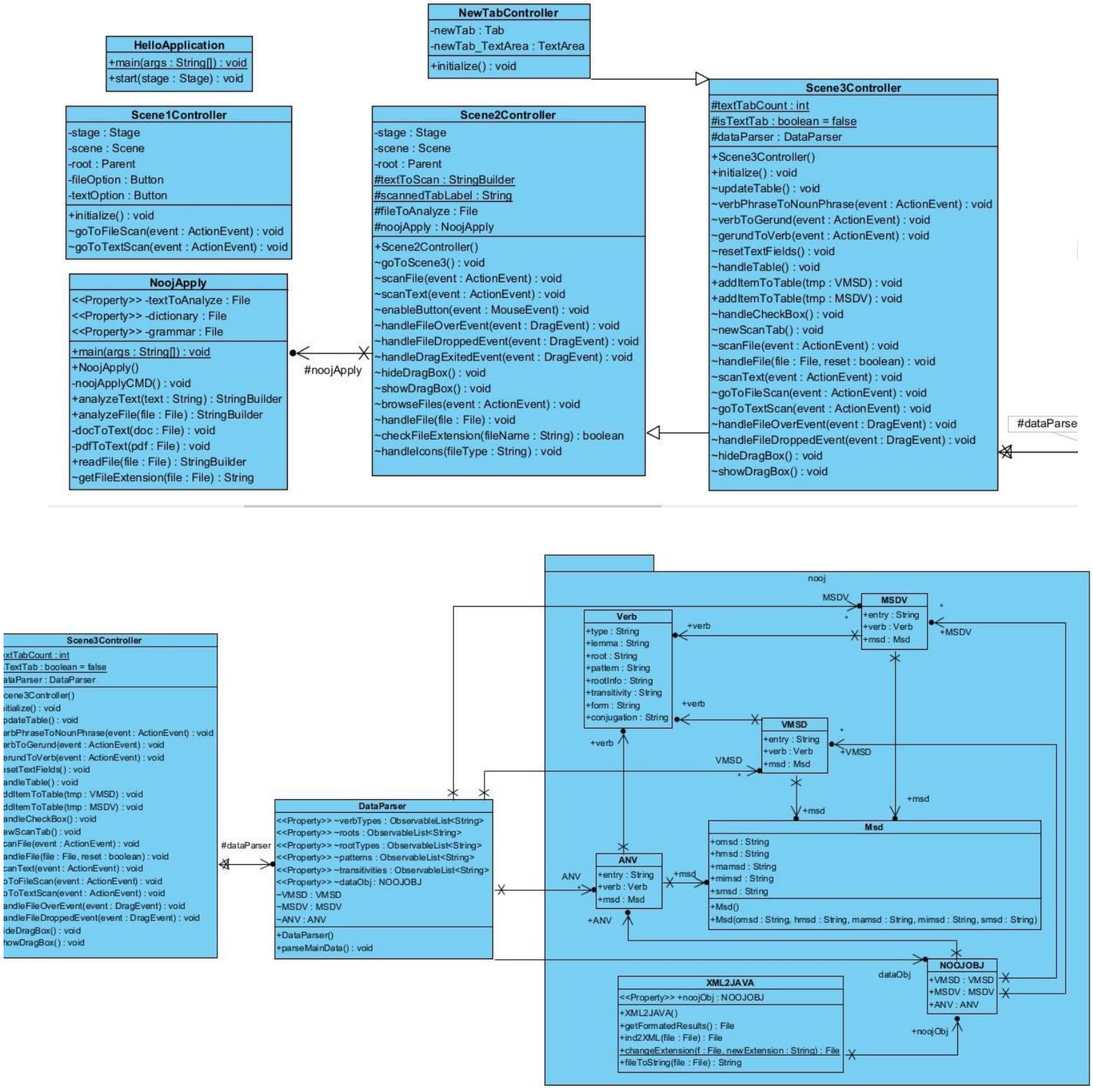

Figure 21 Detailed Class Diagram

3.1.6 Sequence diagrams

File Scan Sequence Diagram

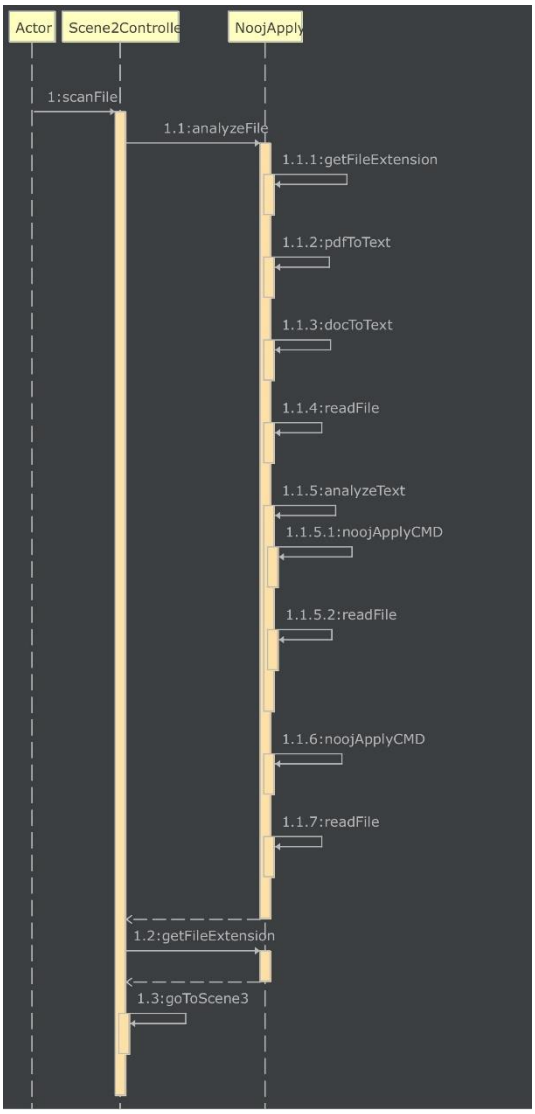

Figure 22 File Scan Sequence Diagram

Text Scan Sequence Diagram

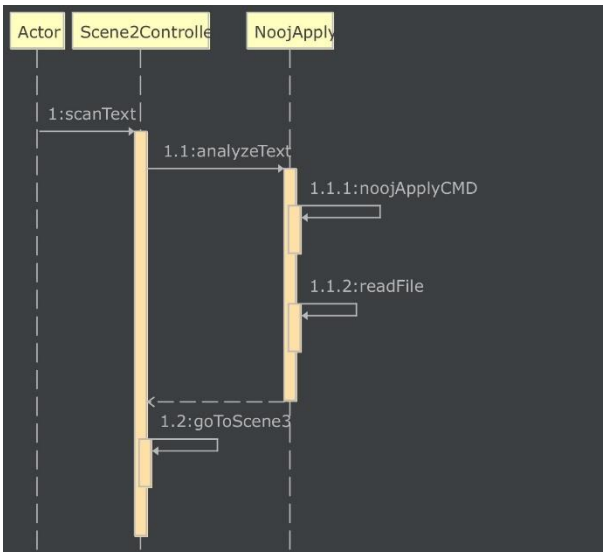

Figure 23 Text Scan Sequence Diagram

Verb to Gerund Sequence Diagram

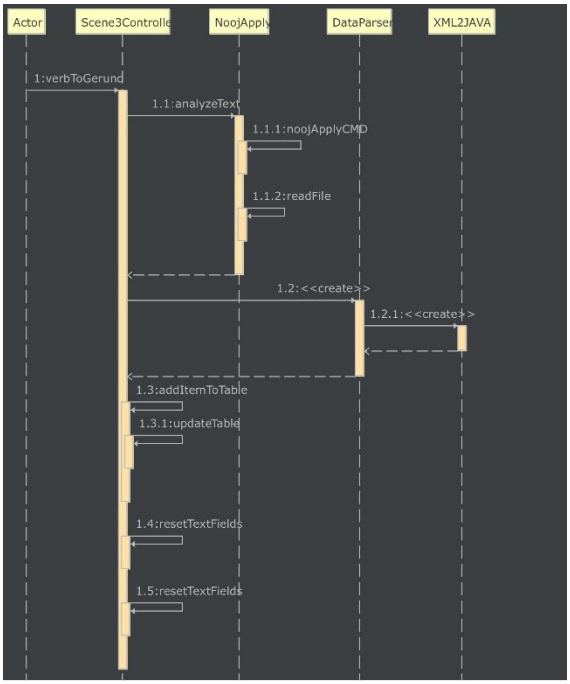

Figure 24 Verb to Gerund Sequence Diagram

Gerund to Verb Sequence Diagram

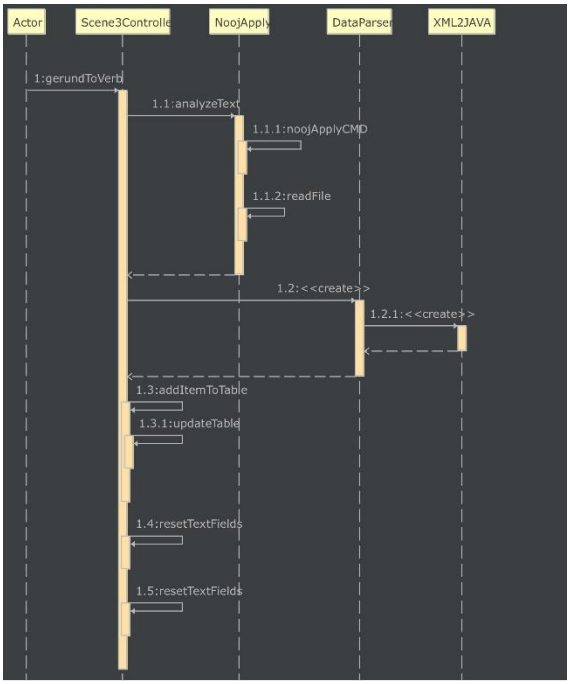

Figure 25 Gerund to Verb Sequence Diagram

Verb Phrase to Noun Phrase

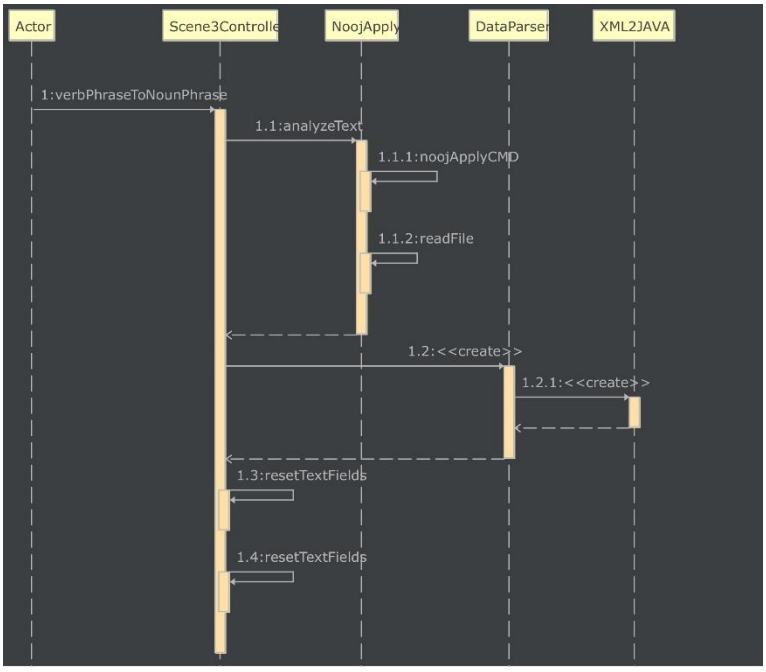

Figure 26 Verb Phrase to Noun Phrase Sequence Diagram

3.2 Application Development & Implementation

In this section, we highlight the development process of the application and its implementation. We discuss briefly the Ui/UX design and move on to applying logic to this user interface.

3.2.1 Application UI/UX Design

In this step, we have a rough idea of how we want our application to look like. We use Gluon Scene Builder, a drag and drop UI designer tool allowing rapid desktop and mobile app development. Scene Builder separates design from logic, allowing us to quickly and easily focus on the UI/UX aspect of application development.

Scene Builder is open source and works with the JavaFX ecosystem – official controls, community projects. Our choice for Scene Builder is mainly for practicalities and facilitating the application development, as we will show in the later sections.

Being open source and supported by a huge community, also gives us a vast amount of information and documentation we could work with.

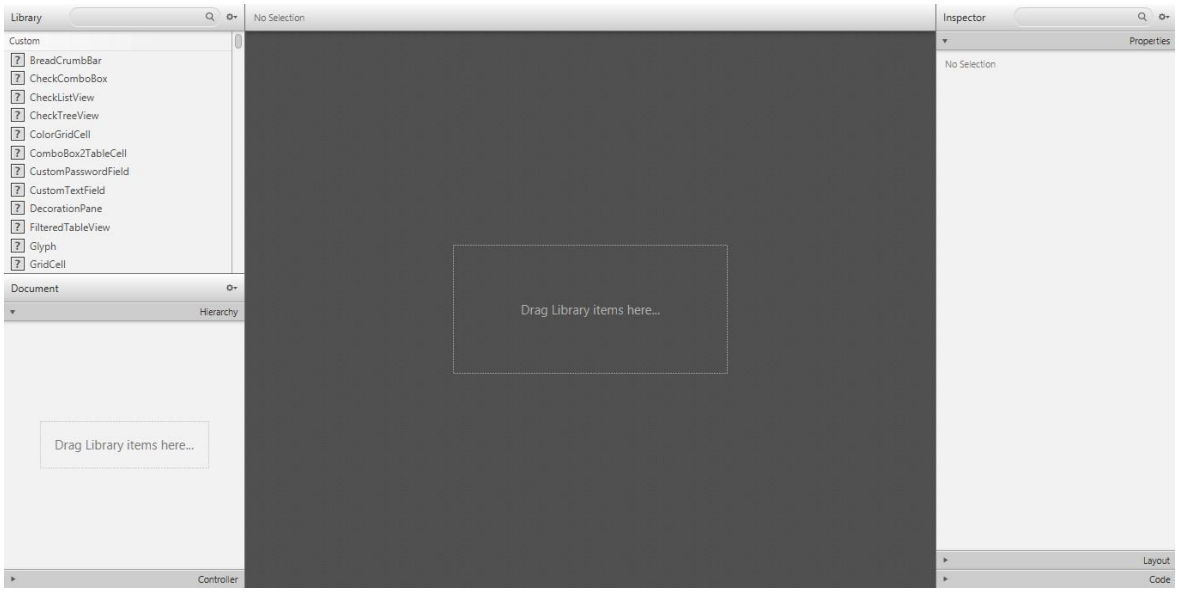

Figure 27 Main Window of Scene Builder

Scene builder provides an easy and modern approach for developing Java Client applications. These applications can run on the JVM or can be converted to platform specific native-images, which have lighting fast startup and takes a fraction of space. Moreover, applications can also be targeted to Android, iOS, and embedded apart from all the desktop environments.

Scene builder all the tools necessary to build these applications including, but not limited to, build tools, IDE plugins, UI library, cloud connectivity, data-binding etc. Figure 20, shows the basic provided elements in Scene Builder.

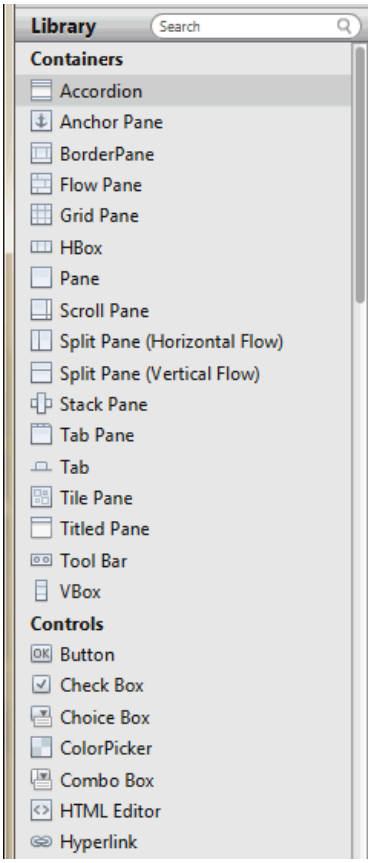

Figure 28 Scene Builder Library Panel

We can use these default elements as they are or combine them to create elements that are more complex. In addition to these elements, Scene Builder allows for the use of third-party libraries, which greatly expands our options of elements.

For instance, we add to Scene Builder the library of UI controls “controlsFX” which we’ll be using in our application. This library has more complex controls than the default ones, like the “CheckComboBox” shown in figure 22.

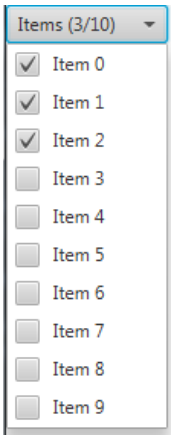

Figure 29 CheckComboBox in the ControlsFX Library

Scene Builder allows us to create “scenes”, which are the different windows of our application. The result is an FXML file that can then be combined with a Java project by binding the UI to the application's logic.

- The FXML file, containing the description of the user interface, is the view.
- The controller is a Java class, optionally implementing the Initializable class, which is declared as the controller for the FXML file.
- The model consists of domain objects, defined on the Java side that can be connected to the view through the controller.

Some of the created scenes for our application:

The first scene (Scene1) represents the first options the user can choose.

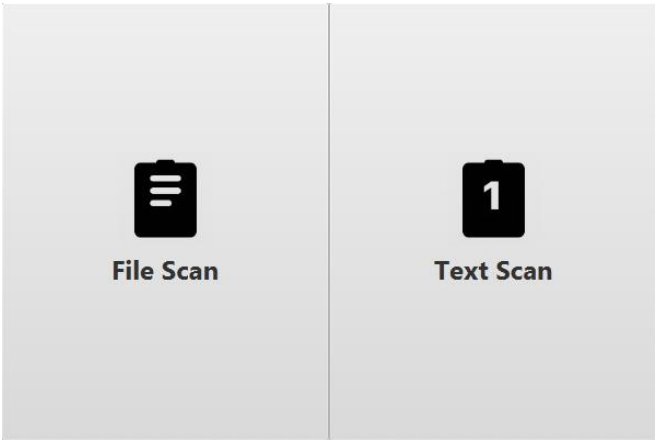

Figure 30 Scene Builder – Scene 1

The second scene (Scene2) represents two options in different tabs the user can use. Each tab representing a different feature of the java application.

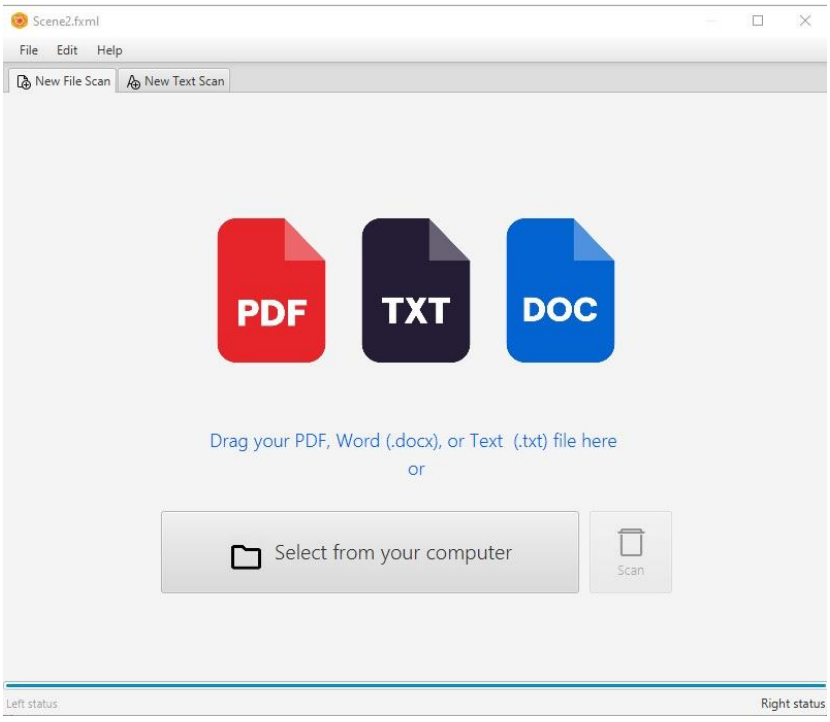

Figure 31 Scene Builder – "File Scan" tab in scene 2

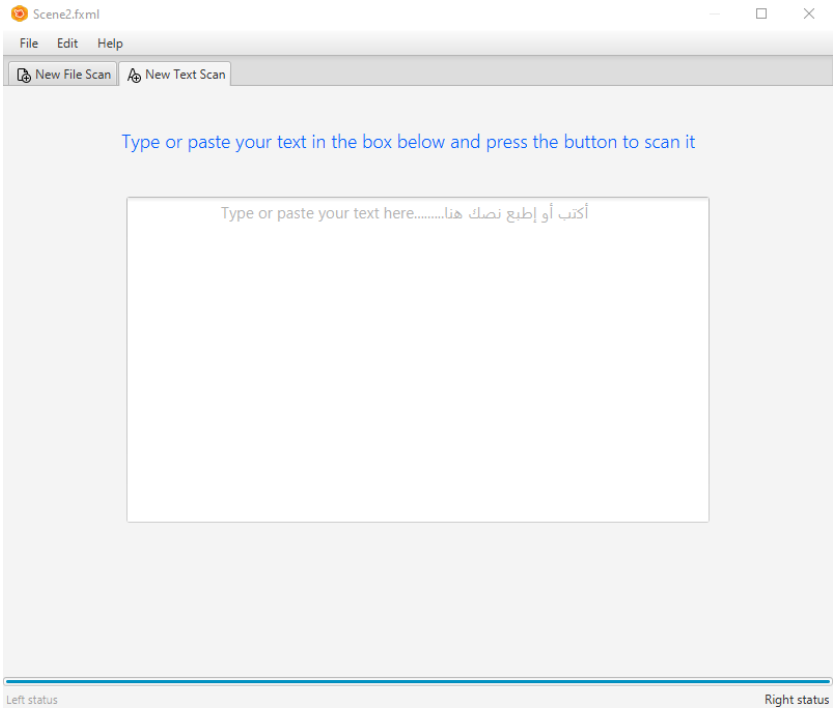

Figure 32 Scene Builder – "Text Scan" tab in scene 2

The third scene (Scene3) represents the principal window of the application. This scene contains most of the features and options the application has to offer. The user could also make different analyses in different tabs, which represents our multi-tasking feature.

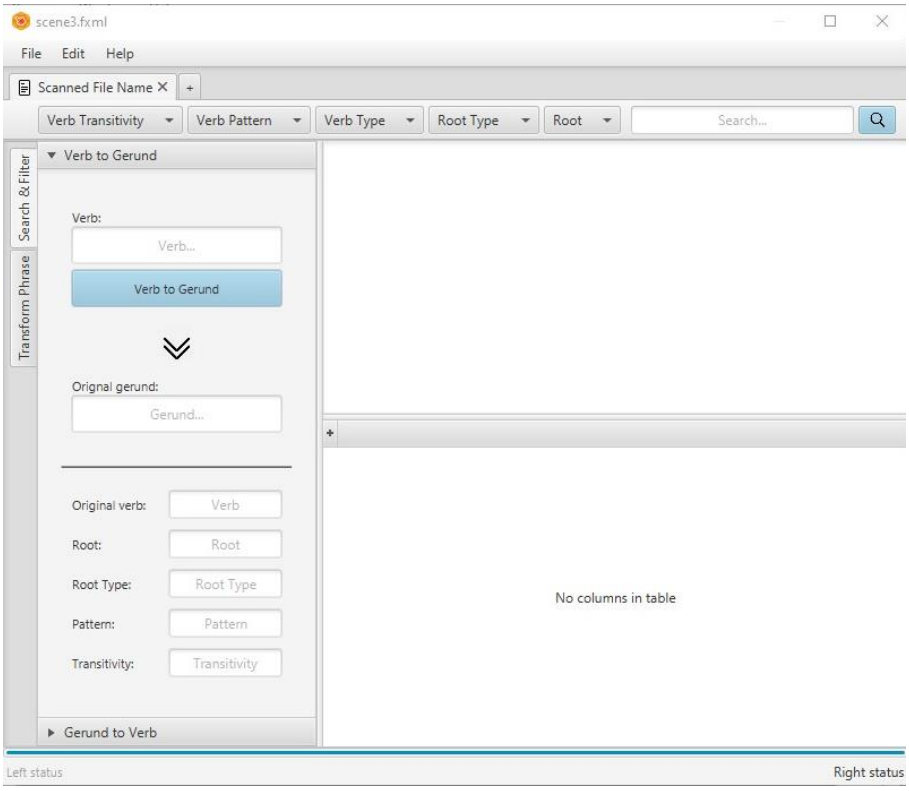

Figure 33 Scene Builder – Scene 3

We now have different scenes that we will be using as reference for our application. In addition, by adding the appropriate id's (fx:id) to each element, adding the JAVAFX library to our project, and compiling these scene into a FXML file (.xml), we could apply logic to any desired element and use them directly in our Java application. We explain more about these features and their uses in later sections.

3.2.2 Application Development

Throughout this project, we use different frameworks, programs and libraries to facilitate and modulate our workload. Some are crucial to our application e.g. Noojapply, while other one could be replaced with faster and more efficient technologies. The various technologies take care of many tasks that we would have taken a lot of time to make from scratch, hence why we rely on many third-party technologies.

This project involves the following technologies:

- **JAVA:** The programming language used.

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need to recompile. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019, Java was one of the most popular programming languages in use according to GitHub, particularly for client–server web applications, with a reported 9 million developers.

- **Intellig IDEA:** The IDE we use to program the application.

IntelliJ IDEA is an integrated development environment (IDE) written in Java for developing computer software written in Java, Kotlin, Groovy, and other JAR based languages. It is developed by JetBrains (formerly known as IntelliJ), and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition. Both can be used for commercial development.

- **Gluon Scene Builder:** The UI designer tool used.

Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding. Users can drag and drop UI components to a work area, modify their properties, apply style sheets, and the FXML code for the layout that they are creating is automatically generated in the background.

- **JAVAFX:** The software platform for creating and delivering the Java application.

JavaFX is a software platform for creating and delivering desktop applications, as well as rich web applications that can run across a wide variety of devices. JavaFX has support for desktop computers and web browsers on Microsoft Windows, Linux, and macOS, as well as mobile devices running iOS and Android.

- **ControlsFX:** This is a JAVAFX UI controls library.

ControlsFX is a library of UI controls and useful API for JavaFX 8.0 and beyond.

- **Noojapply:** Nooj's API. Used to communicate with NooJ.

Noojapply is a command-line program, which can be called either directly from a "shell" script, or from more sophisticated programs written in PERL, C++, JAVA, etc. In our work, we use JAVA. It allows us to apply the given lexical resources (Dictionary with inflectional and derivational grammars) and then the syntactic resources (Syntactic grammar) to each text, annotates them, and then produce the .ind text file, which contains the desired results in an XML format.

- **XML:** The main file format for our UI and the results from Noojapply.

XML (Extensible Markup Language) is a markup language similar to HTML, but without predefined tags to use. Instead, you define your own tags designed specifically for your needs. This is a powerful way to store data in a format that can be stored, searched, and shared

- **JAXB: Jakarta XML Binding.** This converts XML to Java objects.

Jakarta XML Binding (JAXB; formerly Java Architecture for XML Binding) is a software framework that allows Jakarta EE developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to unmarshal XML back into Java objects. In other words, JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure.

- **PdfToText:** CMD program included with NooJ. It converts PDF files to text files.

- **DocTo:** CMD program used to convert Word Document files to text files.

Simple utility for converting a Microsoft Word Document '.doc', Microsoft Excel '.xls' and Microsoft Powerpoint .ppt files to any other supported format such as .txt .csv .rtf .pdf.

3.2.3 Application Implementation

- **File & Text Scan (Gerund/Verb Analysis)**

In this section, we display the Java application and some behind the scenes work. We give a preview of the many features the application offers while also detailing the code involved. The “Main” class starts by launching our application and building Scene 1. The first thing that appears when we run the Java application id the Home Screen:

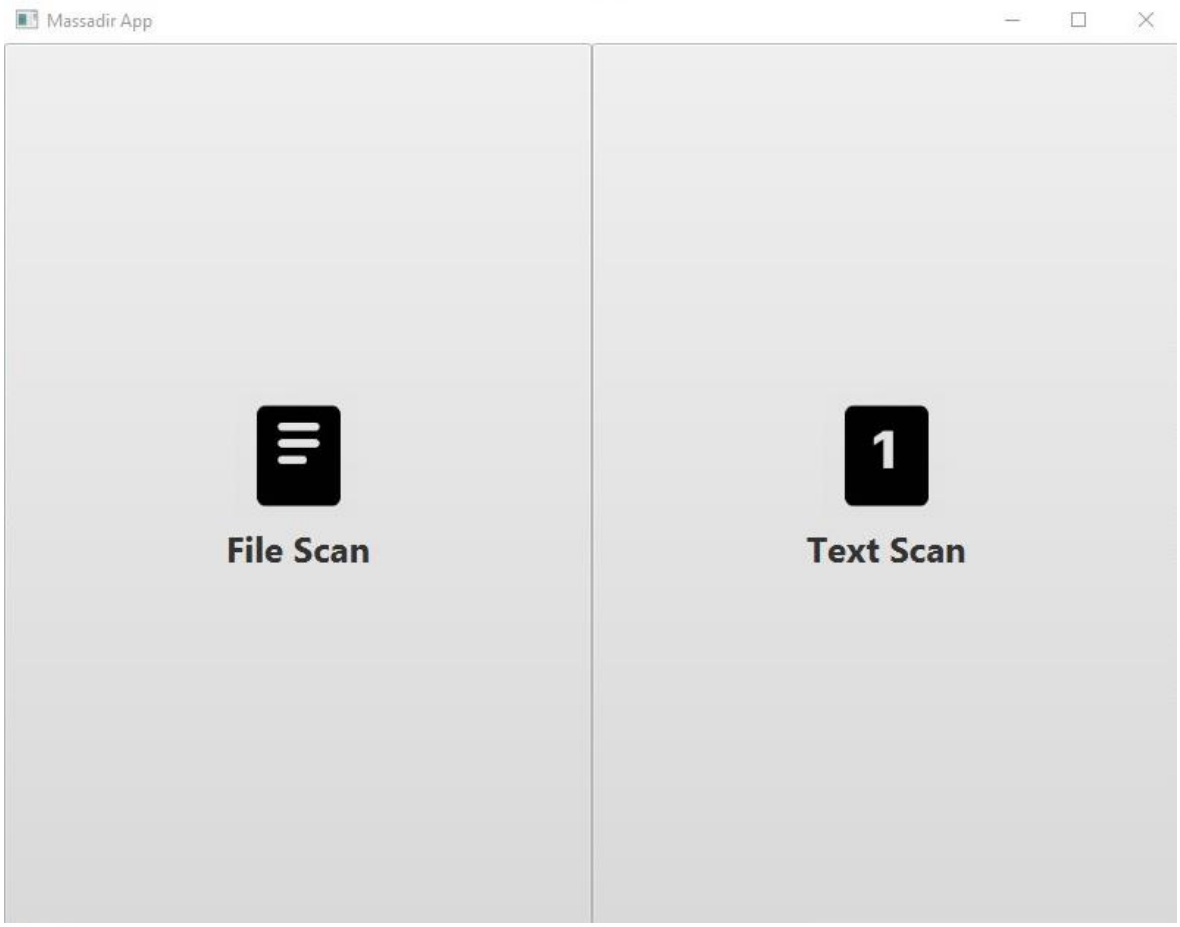

Figure 34 Java Application - Home Screen

After choosing one of the options, the application calls the next scene (Scene2) and selects the corresponding tab to show first. The user can then choose to scan a file or scan an input text. Here we display the Scan File tab.

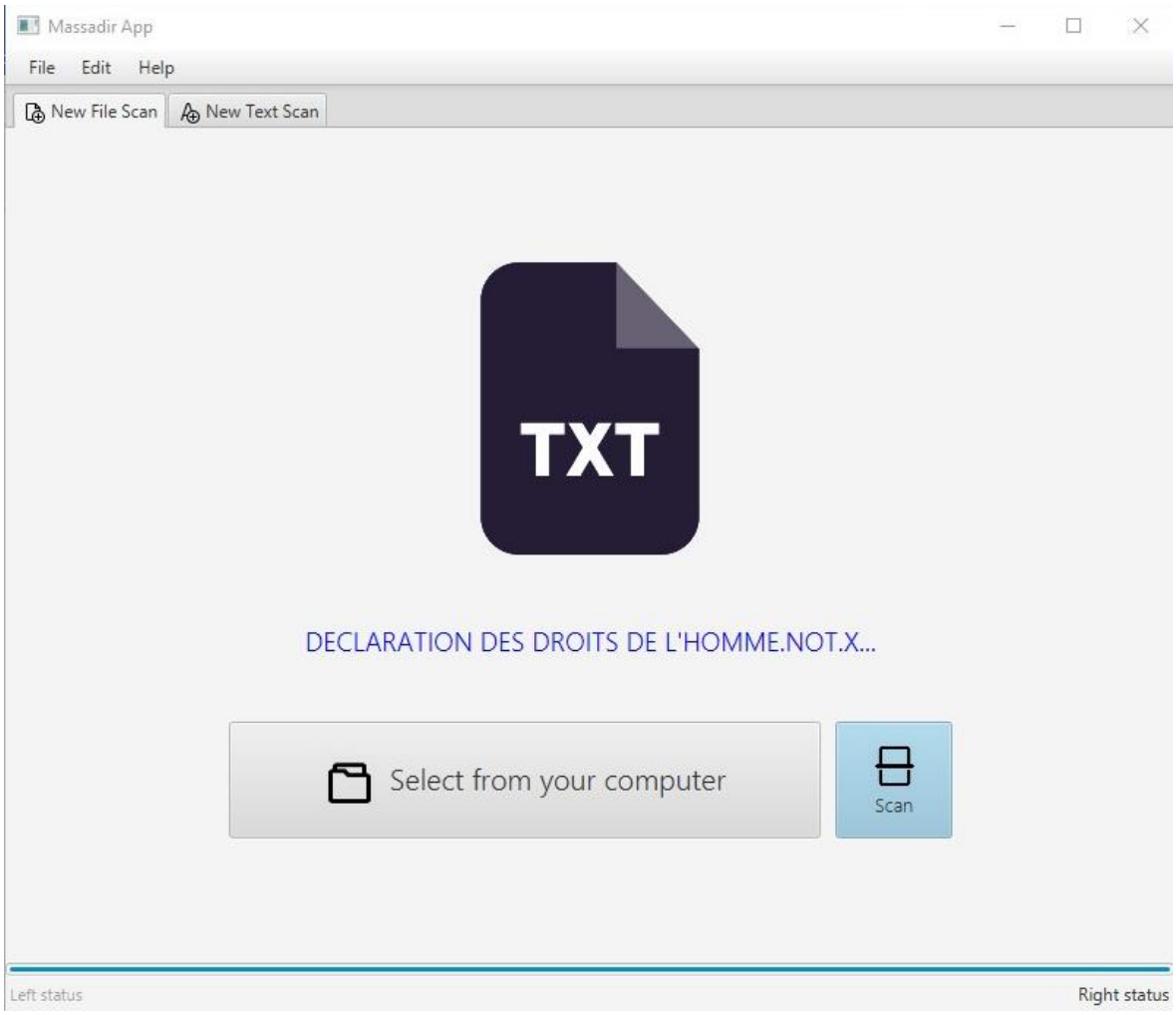

Figure 35 Java Application - File Scan

Here the user can drag & drop his file or chose it from the file browser. The supported files are text files (.txt), PDF files (.pdf), and Word Documents (.docx).

If the user choses a text file (.txt) then the application checks that file and if all the tests are valid, it sends the files content into “textToAnalyze.txt” file, located inside the Noojapply directory so that Noojapply would have an easy access to the text once we fire up the command for it to analyse the text.

Otherwise, if the user choses a PDF file (.pdf) or a Word Document (.docx), then the application checks that file and if all the tests are valid it inputs the file to the respective converter (pdfToText for .pdf) and (docTo for .docx) to get a new text file (.txt) that replaces the “textToAnalyze.txt” file, located inside the Noojapply .

In the Scan File tab, the user can input the Arabic text to be analysed.

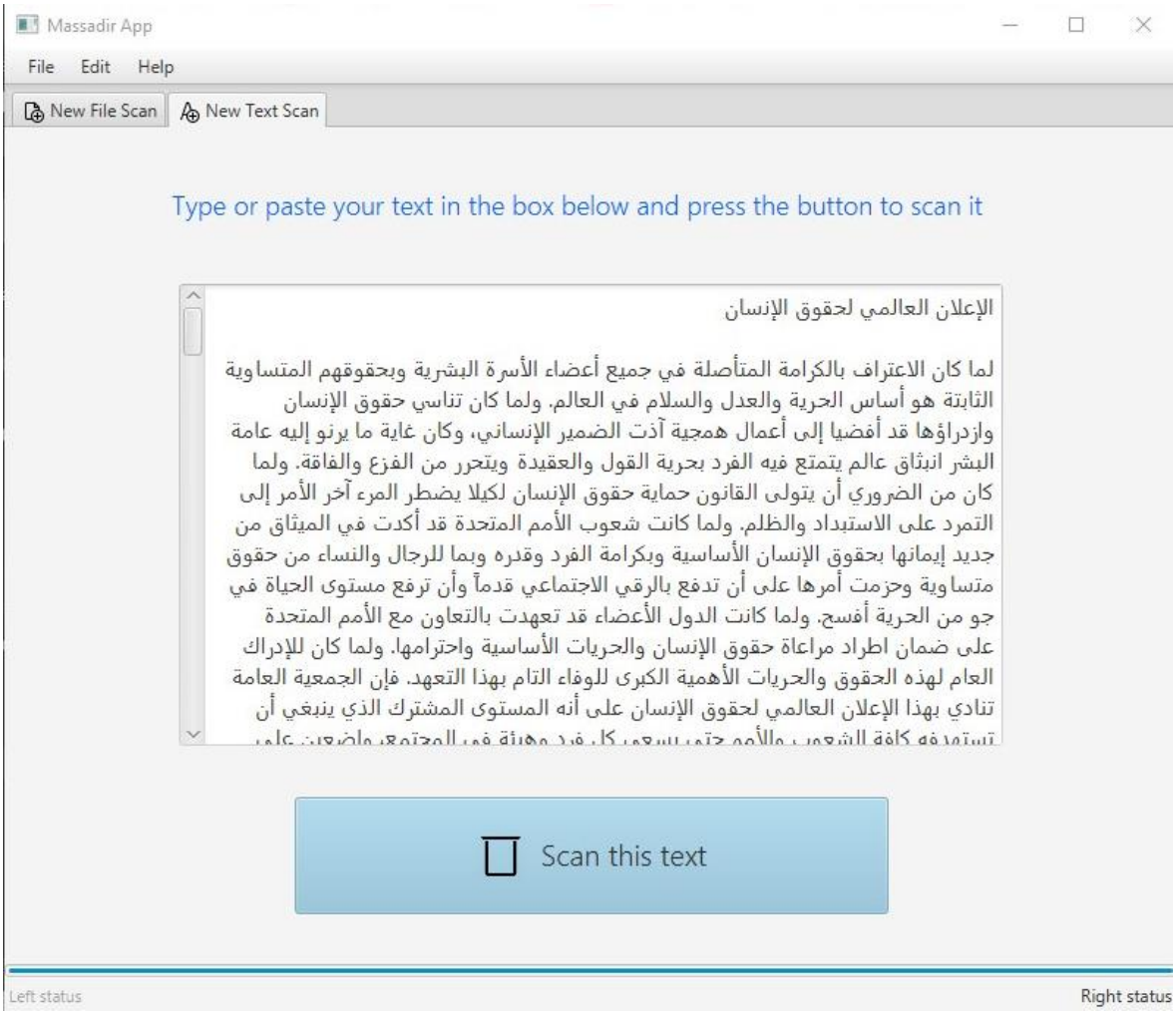

Figure 36 Java Application – Scan Text

This text is then moved inside the “textToAnalyze.txt” file, located inside the Noojapply directory.

After pressing the “Scan” button in either tabs, the application creates a NoojApply object that runs the following command in the windows command line CMD:

```
noojapply ar result.ind bigdic.nod A_grammar.sft textToAnalyze.txt
```

This command runs noojapply.exe and feeds it the required linguistic resources already found inside its directory in addition to “textToAnalyze.txt”, the file that , now, contains our inputted text. Noojapply then uses the linguistic resources (Dictionary and Grammars) to analyse the text and returns an annotated index file (.ind), which is our results file. After that is done, the application calls for the next scene (Scene3).

This Figure 37 represents the main scene, where the user can use all the features our application offers.

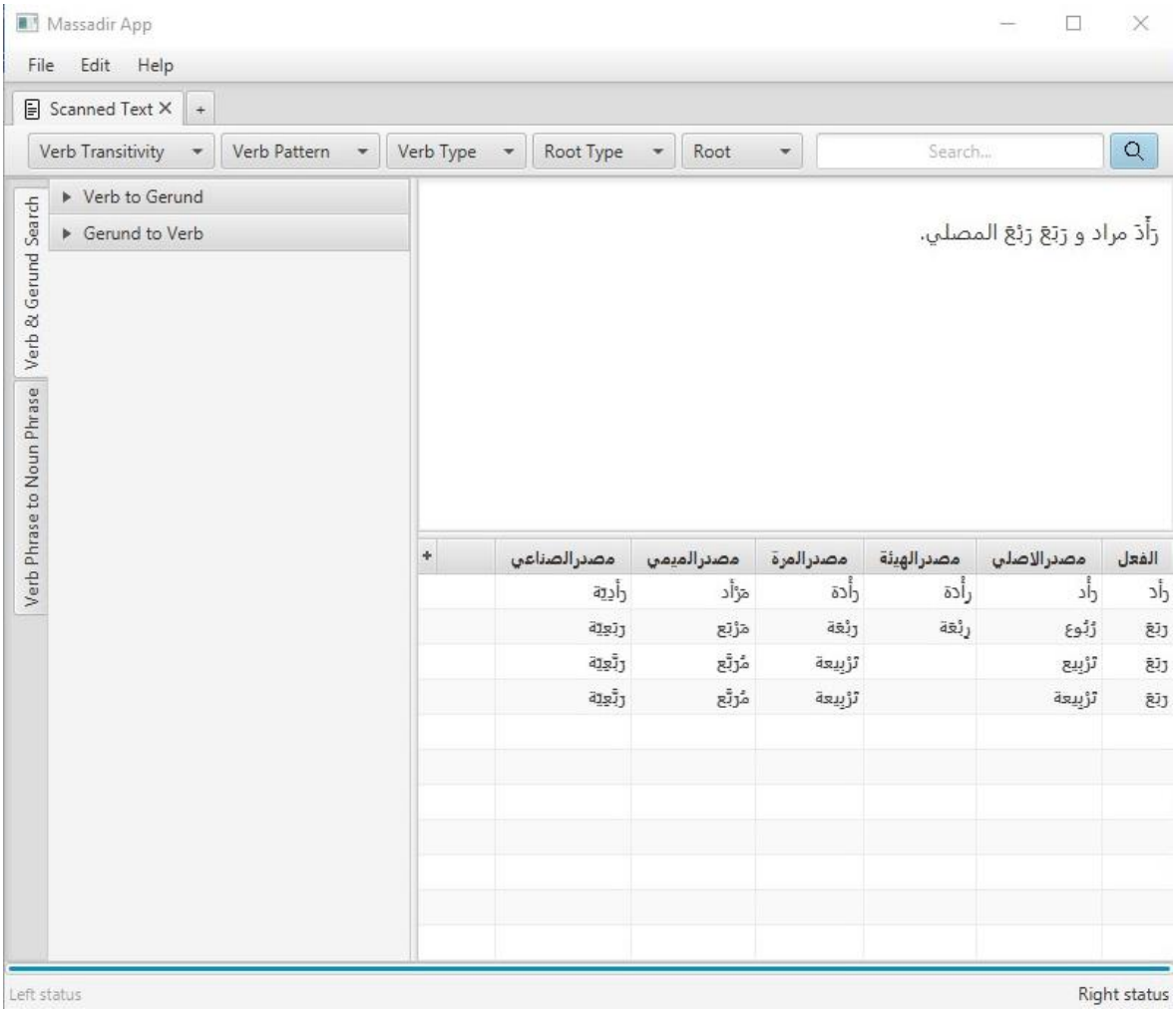

Figure 37 Java Application – Main Scene

The Analysed text has been returned from Noojapply as an (.ind) index file. The Object ind2XML formats this file by adding the "<NoojOBJ>" element as a root and converts it into an XML file (.xml). Then it uses JAXB methods to unmarshal the XML into a Java object of type "NOOJOBJ", which holds 3 lists of the object types MSDV, VMSD, ANV. These types have already their annotated classes by JAXB. As an example, Figure 38 shows the XML-annotated NOOJOBJ class.

```java
import jakarta.xml.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

10 usages
@XmlRootElement(name = "NOOJOBJ")
@XmlType(propOrder = {"VMSD", "MSDV", "ANV"})
@XmlAccessorType(XmlAccessType.FIELD)
public class NOOJOBJ {

    11 usages
    @XmlElement(name = "VMSD")
    public List<VMSD> VMSD ;

    11 usages
    @XmlElement(name = "MSDV")
    public List<MSDV> MSDV;

    8 usages
    @XmlElement(name = "ANV")
    public List<ANV> ANV;

}
```

Figure 38 XML-Annotated NOOJOBJ Java Class

These pre-defined annotations are what define the different elements and attributes, so they could be read and mapped to Java objects by JAXB.

These lists (MSDV, VMSD, ANV) take the corresponding elements and values from the XML file thanks to JAXB. These lists of Java classes store their values based on the type of XML element, which was described earlier as the annotation by the syntactical grammar (A_grammar.nof) or it might be thought of as the path taken in this grammar.

These Objects take the mapped values from the results XML file corresponding to the correct element. Figure 38 shows the value that one of these objects takes.

```java
package com.massadirapplication.massadirappfx.nooj;

import jakarta.xml.bind.annotation.*;

6 usages
@XmlRootElement(name = "VMSD")
@XmlType(propOrder = { "entry", "verb", "msd"})
@XmlAccessorType(XmlAccessType.FIELD)
public class VMSD {

    2 usages
    @XmlElement(name = "entry")
    public String entry;

    @XmlElement(name = "verb")
    public Verb verb;

    11 usages
    @XmlElement(name = "msd")
    public Msd msd;
}
```

Figure 39 XML-Annotated VMSD Java Class

The Msd and Verb are also Java classes we defined to help store their values. Fig. 40.

```java
package com.massadirapplication.massadirappfx.nooj;

import jakarta.xml.bind.annotation.*;

3 usages
@XmlRootElement(name = "verb")
@XmlType(propOrder = {"type", "lemma", "root", "pattern", "rootInfo", "transitivity", "form", "conjugation"})
@XmlAccessorType(XmlAccessType.FIELD)
public class Verb {
    3 usages
    @XmlElement(name = "type")
    public String type;
    6 usages
    @XmlElement(name = "lemma")
    public String lemma;
    @XmlElement(name = "root")
    public String root;
    6 usages
    @XmlElement(name = "pattern")
    public String pattern;
    5 usages
    @XmlElement(name = "rootInfo")
    public String rootInfo;
    6 usages
    @XmlElement(name = "transitivity")
    public String transitivity;
```

Figure 40 XML-Annotated Verb Java Class

Once the Java objects are mapped and their values are extracted from the XML file (as Figure 41 shows), the application parses this data into the table in the main scene (Scene3).

```
<NOOJOBJ>
    <VMSD>
        <entry>رَأَبَ</entry>
        <verb>
            <type>Tri</type>
            <lemma>رَأَبَ</lemma>
            <root>ر أ ب</root>
            <pattern>فَعَلَ</pattern>
            <rootInfo>CHC</rootInfo>
            <transitivity>Tr</transitivity>
            <form>VNN</form>
            <conjugation>aa</conjugation>
        </verb>
        <msd>
            <omsd>رَأْب</omsd>
            <hmsd>رَأْبَة</hmsd>
            <mamsd>رَأْبَة</mamsd>
            <mimsd>مَرْأَب</mimsd>
            <smsd>رَأْبِيّة</smsd>
        </msd>
    </VMSD>
</NOOJOBJ>
```

Figure 41 XML Annotated Results File

After the mapping of the extracted values from the text in the table view, the application displays the verbs and their different gerund types. As shown in Figure 42.

الفعل	مصدرالاصلي	مصدرالهيئة	مصدرالمرة	مصدرالميمي	مصدرالصناعي
تراءد	تراؤُد		تراؤُدة	مُتراءد	تراءدِية
تراءد	تراؤُدة		تراؤُدة	مُتراءد	تراءدِية
تَرأَّد	تَرْئِيد		تَرْئِيدة	مُتَرأِّد	تَرأِّدِية
تَرأَّد	تَرْئِيدة		تَرْئِيدة	مُتَرأِّد	تَرأِّدِية

Figure 42 Java Application – Table View

- **Verb to Gerund Conversion (& vice-versa)**

The side panel of the application gives the user the ability to convert a gerund to a verb or extract the different gerunds from a verb. As shown in Figure 43.

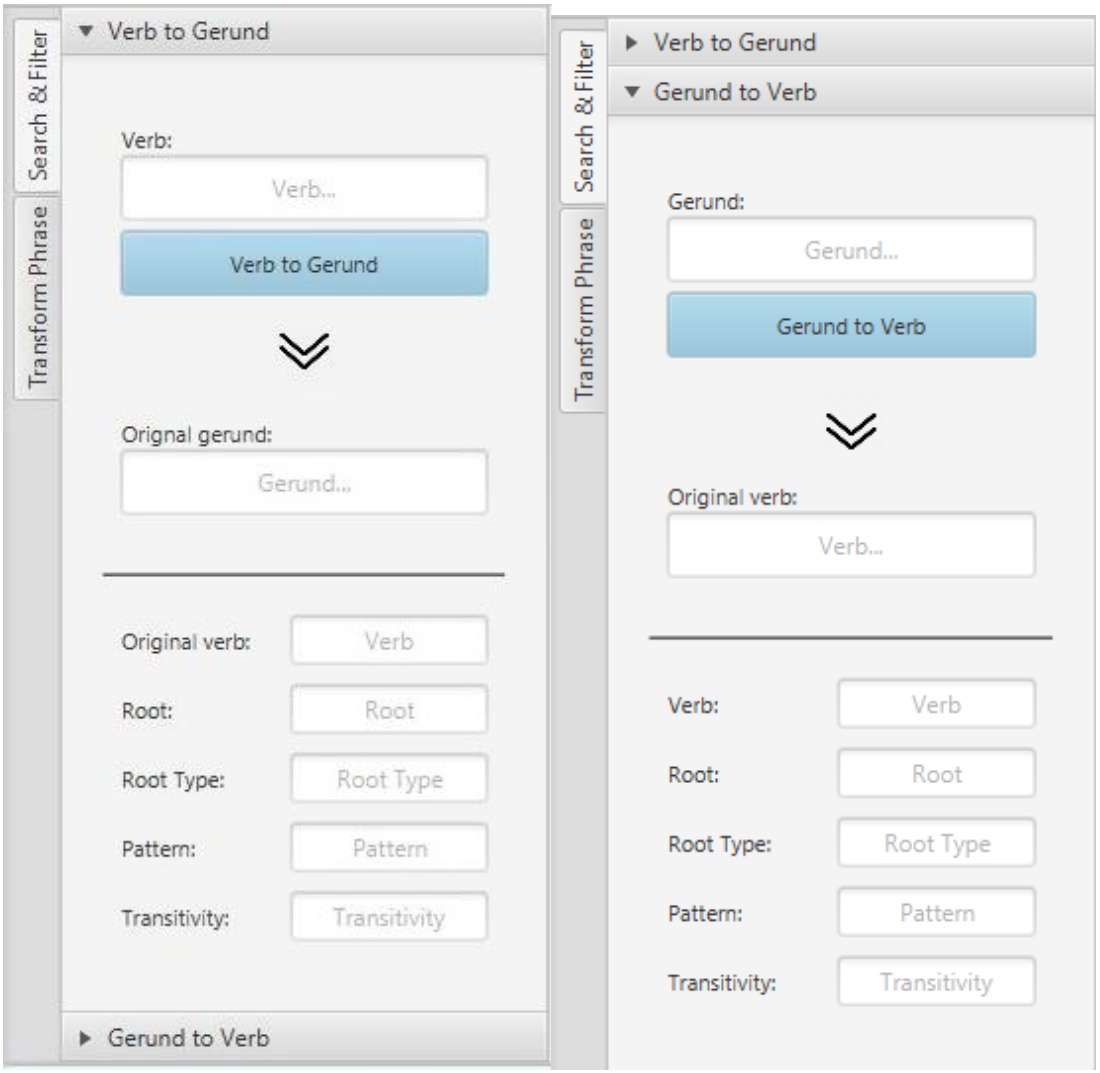

Figure 43 Java Application – Verb & Gerund Conversion

By inputting either a verb or gerund, depending on which conversion type, the application follows the same Text Scan sequence to extract either the verb or gerund. In addition to the resulting verb or gerund, the application also displays the linguistic features of the verb in both cases. In the case of verb to gerund conversion, the original gerund is displayed in the gerund box, however, it also adds a row to the table containing all other types of gerunds.

Figure 44 shows an example of the verb to gerund conversion

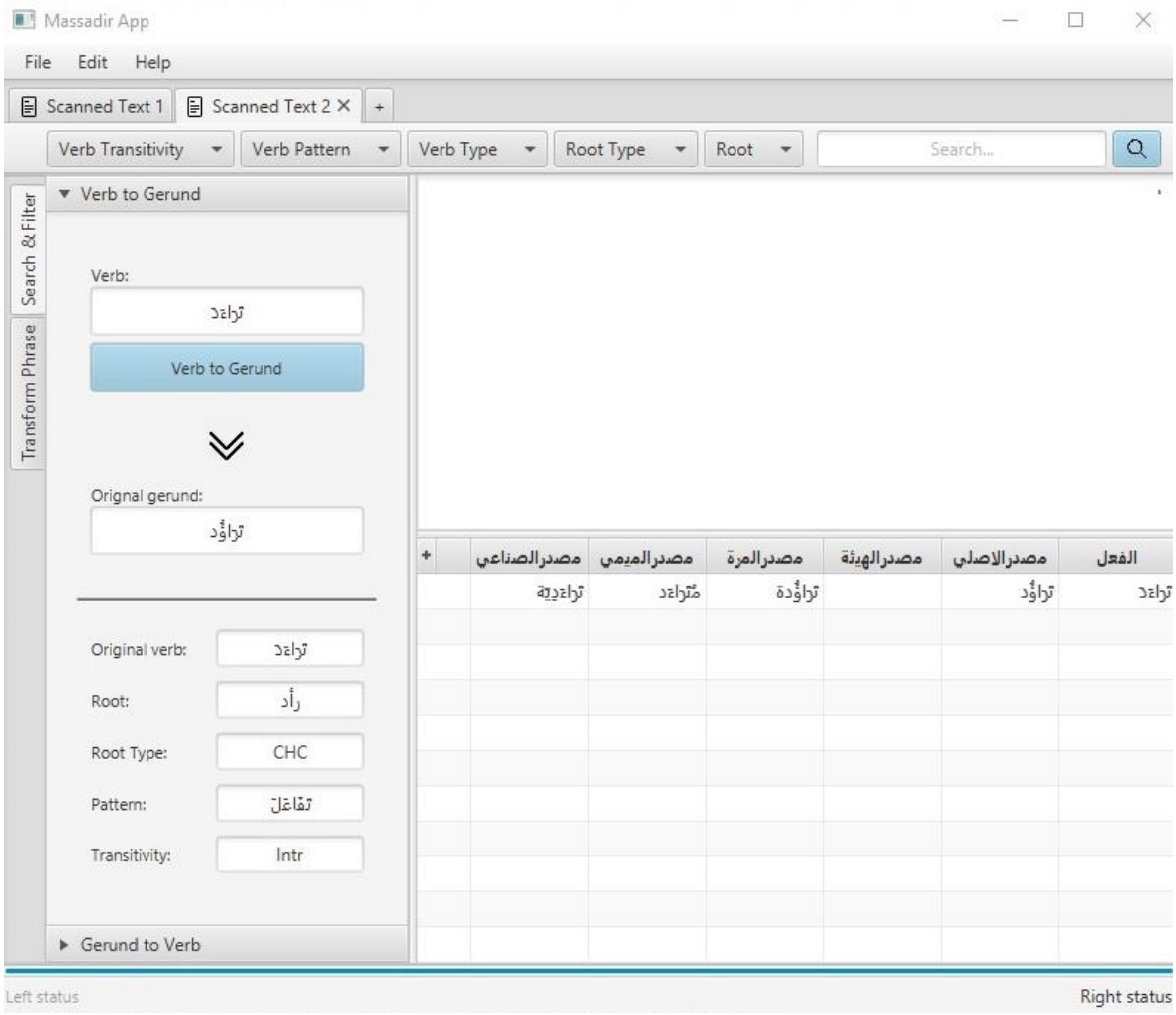

Figure 44 Java Application – Example of Verb to Gerund conversion

- **Verb Phrase to Noun Phrase**

This process takes the inputted phrase looks for the "ان" and the verb in the subjunctive case following it in the string using a regular expression and removes them. After that, the Noojapply process runs again and extracts the one-time gerund (Marra), stores it, and concatenates the string back together to create the verb phrase. Then the application displays the data In addition to some linguistic information about the inputted verb.

Figure 45 shows an example of this process.

Figure 45 Java Application – Example of Verb Phrase to Noun Phrase Conversion

- **Miscellaneous Features**

Multiple tabs support: The user is encouraged to do as many analyses as possible. Thus, we provide an easy-to-use and familiar option, which is to create and open new tabs as Figure 46 shows.

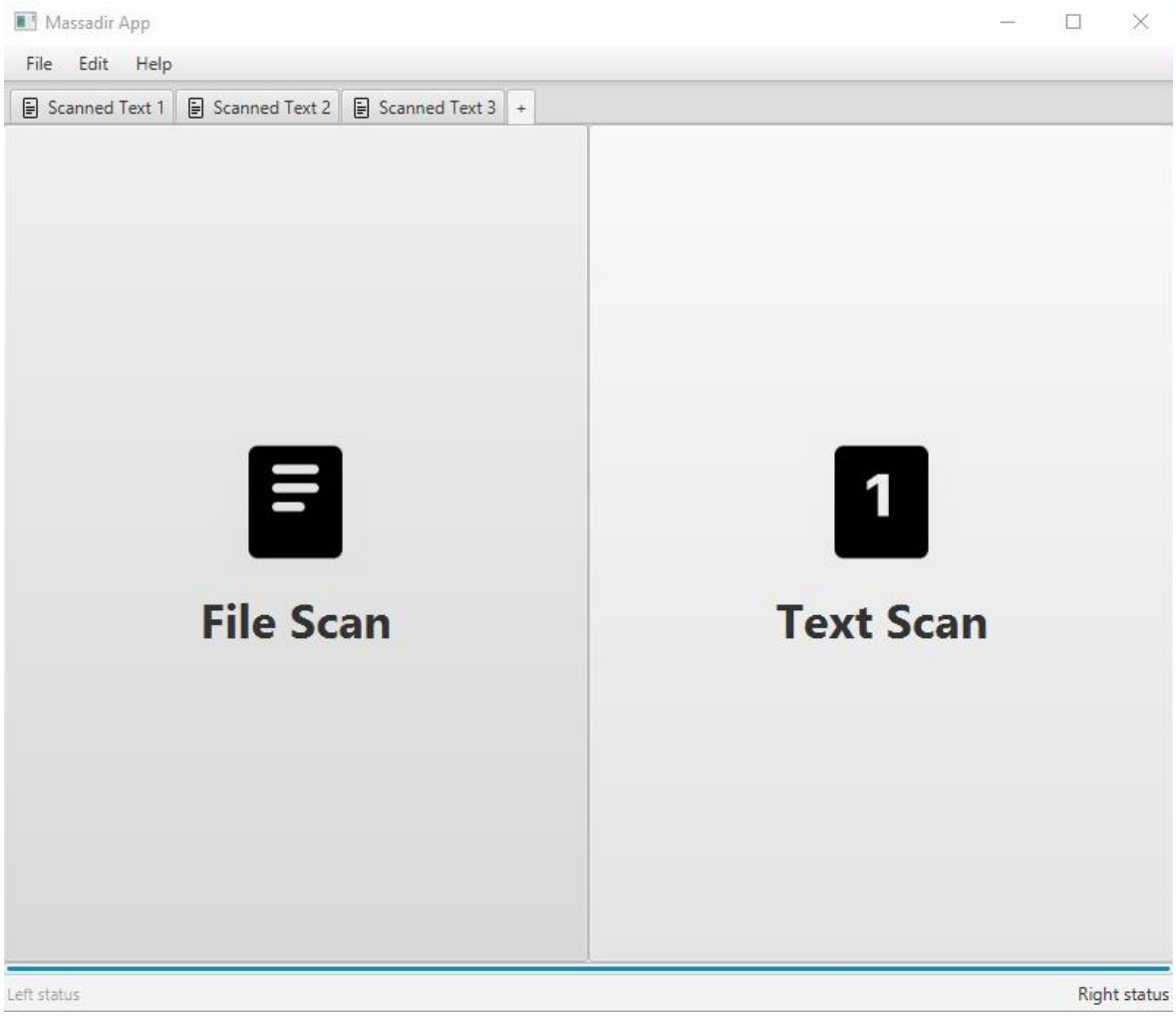

Figure 46 Java Application – Multiple Tab Support

Search & Filters support: The application offers the ability to search for words in the displayed text in the table view. The filters are gathered in the first initialization of the table (first parsing data into it), and mapped into the CheckedComboBoxes, which allows the application to handle and update the table view once any changes have been done or the filters have been changed. Figure 47 shows the search & filter control bar.

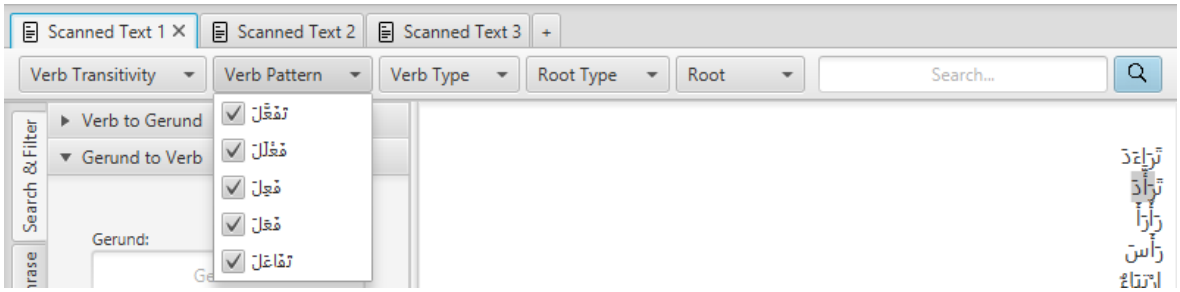

Figure 47 Java Application – Search & Filters Support

Conclusions & Perspectives

In this report, we have developed a Java application that helps users to learn the Arabic language. We have used Arabic verbs and gerunds resources based on the root and pattern approach. We have also used these resources in the NooJ platform, which gives a detailed annotation file after making a linguistic analysis of the given entries. We have used these annotations in our application to retrieve the linguistic features of any verb/gerund occurring in the inputted text. The application also examines the rule/rules that were applied to the verbs and gerunds. Behind this process, a deep linguistic study was made, which yielded many different rules that were used in our grammar. Each rule consists of testing the initial form of linguistic features to get the derivational forms and the inflectional forms. Rather than storing these rules in databases, the application tests the linguistic features on the entry to extract the desired derivational form, inflectional form, and linguistic data.

The perspectives emerging from this work are to complete the Java application by finishing the verb part with other kinds of Arabic verbs since we have the “Arabic verb starting with (R/ر) resource. We are also going to add more rules to the grammar for the verbs and gerunds. This project has given me a great first experience in the field of Natural Language Processing and a whole new different point of view on the complex and difficult aspects of the Arabic language that learners and researchers face when trying to deeply understand the language. With that, I feel huge respect for linguistics and linguists that are trying to better our understanding of the language and to make it more accessible for others by tackling projects like this one.

References

- "Formalizing Arabic inflectional and derivational verbs based on root and pattern approach using NooJ platform." by: Mbarki, S., Mourchid, M., Silberztein, M. (eds.) NooJ 2017. CCIS, vol. 811, pp. 52–65. Springer, Cham(2018).
- "Standard Arabic Verbs Inflections Using Nooj Platform" by: Mohammed Mourchid, Ilham Blanchete, and Abdelaziz Mouloudi. In International Journal on Natural Language Computing · February 2017.
- "Arabic Learning Application to Enhance the Educational Process in Moroccan Mid-High Stage Using NooJ Platform" by: Ilham Blanchete, Mohammed Mourchid, Samir Mbarki, and Abdelaziz Mouloudi. In Formalizing Natural Languages with NooJ 2019 and Its Natural Language Processing Applications.
- "The Treatment of Gerund Forms for Arabic Nouns with LKB System" by Samia Ben Ismail, Sirine Boukédi and Kais Haddar. Sousse University, Miracl laboratory, National Engineering School, Gabes University, Sfax University,Tunisia.
- Silberztein, M.: Nooj Manual. www.nooj-association.org (01/07/2022)
- NooJ: https://site-nooj.blogspot.com/p/arabic-tutorials.html (01/07/2022)
- ControlsFX: https://github.com/controlsfx/controlsfx/wiki/ControlsFX-Features (01/07/2022)
- JAVAFX: https://wiki.openjdk.org/display/OpenJFX/Main (01/07/2022)
- Gluon Scene Builder: https://gluonhq.com/products/scene-builder/ (01/07/2022)
- DocTo: https://github.com/tobya/DocTo (01/07/2022)
- JAXB: https://mkyong.com/java/jaxb-hello-world-example/ (01/07/2022)
- Java Documentation: https://javadoc.io/ (01/07/2022)
- Arabic Dictionary "المعجم الوسيط": https://al-maktaba.org/book/7028 (01/07/2022)