JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGY
http://e-journal.uum.edu.my/index.php/jict

How to cite this article:
AbuSafiya, M. (2021). Automation of Quranic readings gathering process. *Journal of Information and Communication Technology, 20*(2), 135-169.https://doi.org/10.32890/jict2021.20.2.2

Automation of Quranic Readings Gathering Process

Majed AbuSafiya
Department of Software Engineering
Al-Ahliyya Amman University, Jordan

majedabusafiya@gmail.com

Received: 14/5/2020 Revised: 30/8/2020 Accepted: 7/9/2020 Published: 23/2/2021

ABSTRACT

Quran has been the focus of many computer science and computational linguistics researchers. However, insufficient contribution is directed towards Quranic readings. This field focuses on the different ways in which Quranic words can be recited. Quranic readings gathering is a recitation process that is well-known by the specialists in this field. In this process, all the possible ways of reading a verse are covered in a compact and well-structured manner. To the best of the author's knowledge, no computational automation has been cited for this process in literature. In this paper, this process was algorithmically formulated and a developed software system was presented to automate this process. This system took Quranic verse texts as input and generated a text file containing the detailed gathering of the readings of this verse as done by a specialised reciter. The system was applied to the verses of Surah Al-Baqarah and the output text was carefully validated against authentic books in this field. This system can be the basis of software systems for teaching and vocal construction of Quranic readings gathering. It would be useful for students who are interested in Quranic readings.

Keywords: Algorithm, computational linguistics, Quran, Quranic readings gathering, tree.

INTRODUCTION

Quran is the holy book of Muslims. No book in history has ever taken the attention and care that were given to this miraculous book. The scientific work - to serve this book - touched various fields of computer science and information technology. This includes searching (Larbi, 2013; Mohamed & Shokry, 2020), information extraction (Yauri et al., 2013) and mining (Majdalawieh et al., 2017), software systems to facilitate its learning, teaching (AlKhatib et al., 2020; Abdou & Rashwan, 2014; Menacer et al., 2013) and memorisation (Bin Musa et al., 2018; Rafi et al., 2019), recognising its textual (Alotaibi et al., 2018) and vocal forms (Qayyum et al., 2108), studying its syntactical structures (AbuZeina & Alsaheb, 2013; Khadangi et al., 2018), building databases and websites to make it available in the web (Boussenan, 2013; Priatna et al., 2020) and in mobile applications (Abdullah et al., 2019; Elmarhomy, 2020), securing the integrity of its electronic versions against alteration (Hakak et al., 2017), studying the spiritual and mental effects of its recitation (Khan et al., 2010), validating the correctness of its vocal recitation (Rashid et al., 2013) and many other fields.

In the context of Quranic readings, there has been some work towards teaching Quranic readings (Ishak et al., 2016). Mahmoud and Hassan (2013) developed a software system where the different readings of a Quranic verse were retrieved from a database with explanations but without gathering. The authors of the present study could not find any work in the automation of Quranic gathering process. One reason for that could be the nature of the Quranic readings field. It is one of the Quranic scientific fields that are not widely known. Among specialists in Quranic studies, only a very small fraction chooses to specialise in this field. This field is vast and not easy to master. The gathering of Quranic recitation is deemed even harder.

Many books that document the readings gathering for individual Quranic verses - based on the narration ways of *Tayyebat-Annashr*[1] - can be found in the literature (Abdeen, 2016; Salem, 2003). These books briefly describe the gathering of readings for every Quranic verse without the generation of detailed readings. However, it is found that one book (Tulba & Awadh, 2016) documented the readings gathering of Quranic verse in detail. However, this book - contained in five volumes - covered only less than one-fifth of the Quran. This book was the present study's main reference to validate the gathering that is generated by the proposed system.

Quranic Readings Gathering Process

Many words of the Holy Quran can be read in more than one way. These different ways of reading Quranic words are vocally transferred from one generation to the next. Among very early generations, there are twenty narrators that were well-known for their integrity and competency in Quranic readings. Each of these narrators had learned some of the readings of Quranic words and taught them to others. Every one of them became well-known for his narration. It is believed that all authentic readings of the Quranic verses are found in one of these twenty narrations.

Students who are interested in learning Quranic readings put effort to master these twenty narrations. A student is assumed to master a narration and is then authorised to teach it if he recites the whole Quran with that narration before an authorised scholar. He repeats this process for each of the twenty narrations. These narrations are learned in a certain order that is known in this field (Figure 1). This order does not reflect the authenticity of the narration. Once a student masters these twenty narrations separately, he becomes qualified to what is known in Arabic as *Jam'e Al-qiraat* (جمع القراءات),which is literally translated to English as *gathering of readings*. In this stage, the student covers all the twenty narrations through one recitation of the Holy Quran. The gathering of readings can be defined as the recitation of Quranic verse (or *ayah* in Arabic) where all the possible readings of its words are covered. From now on, the word *ayah* will be used in this paper to refer to the Quranic verse.

1 *Tayyebat-Annashr* (Ibnu Al-Jazari, 2003) - written seven centuries ago - is a one-thousand-line poem that summarises the ways, rules, and differences among the trustworthy recurrent narrations of Quran. It is widely accepted to be the most comprehensive and authentic reference in this context.

Figure 1

The Order of the Twenty Narrators

Rank	1	2	3	4	5	6	7	8	9	10
Narrator	Qaloon	Warsh	Bazzi	Qonbul	Dori-AbuAmr	Soosi	Hisham	Ibn-Thakwan	Shuba	Hafs
Rank	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Narrator	Khalaf	Khallad	Abu-Alharith	Dori-Alkisaee	Eisa	Ibn-Jammaz	Roways	Rawh	Ishaq	Idrees

The Quranic readings gathering approach that this study is interested in is known as *gathering by waqf.* The reciter reads until he stops at some suitable point. He then returns to read what he already read through several passes so that he covers all narrations compactly and without mixing up. In this paper, the process of the Quranic readings gathering process is formally described and automated by a software system. The motivations for this research include:

1. Gathering of readings is a complex process, and even the most competent reciters are subject to slip into mistakes. However, this process is well-defined and thus may be automated. If correctly implemented, a software system may generate an accurate and correct gathering. This gathering can be used as a reference for the interested students.
2. Although many books were written to gather the readings of Quranic *ayahs*, they present the gathering as a summarised description without a detailed generation of gathering. The reasons could be due to the huge size of the text and the amount of effort that are required to document the gathering in detail. A software system will be a better alternative to generate a detailed gathering of Quranic *ayah* readings instead of a manual generation that will be error-prone and expensive in terms of time and effort.
3. This process is performed for the twenty narrations as previously mentioned. With a software system, this study can generalise the process of readings gathering for a smaller number of narrations. The process can also be applied for more than twenty narrations to consider other less authentic narrations of the Quran.
4. Having a software system for Quranic readings opens the door for utilising information technology solutions to serve this honoured science. This software system can be the basis for learning, searching, and vocal construction of Quranic readings.

The methodology that was followed in this research can be summarised as follows: (1) studying the different readings for different narrations - *ayah* by *ayah* - of Surah Al-Baqarah. These readings follow well-defined rules. These rules are formulated and programmed as conditions to be tested against every letter of the text of the input *ayah*. If a condition is satisfied, the corresponding Quranic reading is associated with that letter. The input to the system will be a text file containing the Uthmani text of the *ayah* to gather its readings. (2) Once the different readings of the letters are identified, the *ayah* is scanned to build the *ayah readings tree*, which will be traversed to generate the readings gathering. The output will be written to a text file with a detailed description. (3) The gathering that is generated by the system is tested against trustworthy readings gathering books to validate the generated gathering. The system is continuously adjusted until the generated *ayah* gathering matches the gathering found in the books before advancing to the next *ayah*.

This paper is organised as follows: the second section formalises the readings gathering process. The next section presents an automation solution for the process in the algorithmic level, while the following section discusses several implementation issues. The paper ends with a conclusion and a set of references.

FORMALISING THE QURANIC READINGS GATHERING PROCESS

Before presenting how the gathering is done by specialists, an assumption is made for simplification purposes. This study will consider gathering for the whole *ayah* and not parts of it. This study will assume that the reciter stops at the end of the *ayah*. This assumption is based on two reasons. The first reason is to unify the stopping points to be at the end of the Quranic *ayahs*. This may be justified by the fact that the end of the *ayah* is known to be a stopping point when reciting the Quran. The second reason is that with this assumption, the number of passes of the gathering will be equal to the number of ways the *ayah* (as a whole) can be read for the twenty narrators.

The authors studied the gathering process from specialists in this field and formalised it in a form of an algorithm (Figure 2). The algorithm will be described as followed by a human reciter. The reciter starts reading the *ayah* word by word. *CurrentWord* is the word to be read. Initially, the *currentWord* is the first word of the *ayah*. *StoppingNarrators* set is the set of narrators that still have some readings for the *currentWord* that are not yet covered in the gathering. Every word has its own *stoppingNarrators*, which will change through the course of gathering.

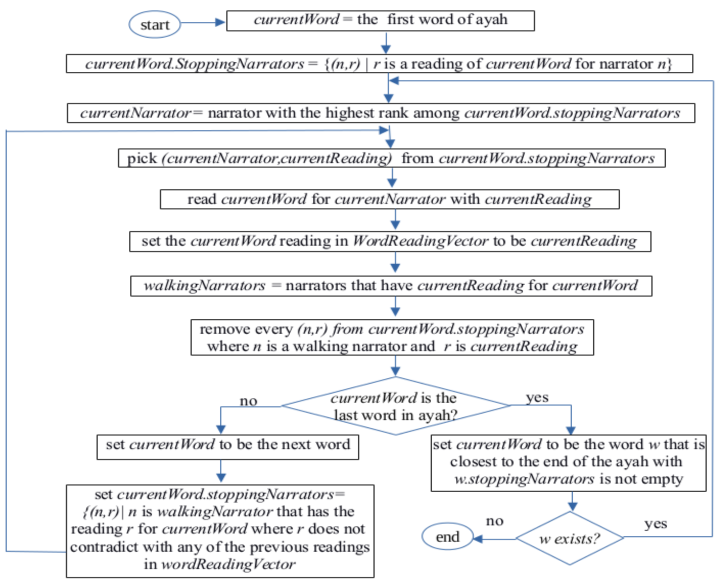
The set of *stoppingNarrators* is represented in the form of a set of ordered pairs. The first coordinate is a *narrator* and the second is a reading for *currentWord* for that *narrator*. For example, assume that n_1 is a narrator that is stopping at *currentWord* and has the readings r_1 and r_2 for *currentWord*. Let n_2 be another narrator that is stopping at *currentWord* and has the readings r_1 and r_3 for *currentWord.* In this case, *stoppingNarrators* set for *currentWord* will be $\{(n_1,r_1), (n_1,r_2), (n_2,r_1), (n_2,r_3)\}$.

Initially, the *stoppingNarrators* for the first word of the *ayah* is $\{(n,r) \mid r$ is a reading for *currentWord* for narrator $n\}$, where all narrators are stopping there with no one stopping anywhere else yet. *CurrentNarrator* is the narrator who is reading for in the current pass. In this paper, *pass* is defined as a reading of the *ayah* starting from some word within the *ayah* and ending by its last word for a given narrator(s). A pass corresponds to one of the narrations of the *ayah* as a whole. *CurrentNarrator* is always the narrator with the highest rank among the stopping narrators at the *currentWord* (Figure 1). The algorithm iterates until all *stoppingNarrators* for all the words of the *ayah* become empty.

Once *currentWord* and *currentNarrator* are set, one pair from *currentWord.stoppingNarrators* is picked with the first coordinate of *currentNarrator*. If there is more than one pair for *currentNarrator,* then the *currentNarrator* has multiple readings for *currentWord*. The reciter will then pick the pair with the reading of the highest priority. There is a certain ordering of the different readings that is known by the reciters. This selected reading will be referred to as *currentReading*. The reciter will read the *currentWord* for *currentNarrator* with *currentReading*. This *currentReading* will be stored in the *currentWord's* place in *wordReadingVector*.

Figure 2

Gathering of Readings Process


start
currentWord = the first word of ayah
currentWord.StoppingNarrators = {(n,r) | r is a reading of currentWord for narrator n}
currentNarrator= narrator with the highest rank among currentWord.stoppingNarrators
pick (currentNarrator,currentReading) from currentWord.stoppingNarrators
read currentWord for currentNarrator with currentReading
set the currentWord reading in WordReadingVector to be currentReading
walkingNarrators = narrators that have currentReading for currentWord
remove every (n,r) from currentWord.stoppingNarrators where n is a walking narrator and r is currentReading
currentWord is the last word in ayah?
no
yes
set currentWord to be the next word
set currentWord.stoppingNarrators= {(n,r)| n is walkingNarrator that has the reading r for currentWord where r does not contradict with any of the previous readings in wordReadingVector
set currentWord to be the word w that is closest to the end of the ayah with w.stoppingNarrators is not empty
w exists?
no
yes
end

WordReadingVector is a vector that stores the last reading that was conducted by the reciter for the words of the *ayah*. It is length equals to the number of the words of the *ayah*. Whenever the reciter reads a word of the *ayah* with some reading, then the corresponding element of that word is updated to be that reading. It will stay as is until that word is recited again during the gathering process, in which the new reading of the word replaces the older. As will be shown later, this vector is needed to check for *contradicting readings*.

WalkingNarrators are the stopping narrators at *currentWord* (including *currentNarrator*) that have *currentReading* as one of their readings for *currentWord*. The *stoppingNarrators* set for *currentWord* is updated by removing every pair (*n,r*) where *n* is a walking narrator and *r* is *currentReading*. The narrators in *currentWord.stoppingNarrators* and *walkingNarrators* are not necessarily disjoint.

Next, a decision needs to be made: if *currentWord* is not the last word in the *ayah*, *currentWord* will be set to be the next word in the *ayah*. *StoppingNarrators* for the new *currentWord* will be the set of pairs *(n,r)* where *n* is a narrator from *walkingNarrators* set and *r* is a reading for the new *currentWord* for narrator *n.* In addition to *r* being one of the readings for *n*, *r* should satisfy another condition: the reading *r* should not contradict (for narrator *n*) with any of the readings (of the current pass) of the words existing before *currentWord* in the *ayah*. This condition is checked by examining the readings of the words that exist before the *currentWord* location in *wordReadingVector*. If a contradiction is found, then *(n,r)* will not be added to the *stoppingNarrators* of the new *currentWord*. These contradictions are assumed to be known by the reciter. The following section of example will show how this constraint is checked.

If *currentWord* is the last word of the *ayah*, then the current pass is ended. In this case, the *currentWord* is set to be the word that is closest to the end of the *ayah* and has non-empty *stopping narrators* set. If such a word exists, the algorithm iterates and a new pass starts; otherwise, the algorithm terminates.

Example of Gathering of Readings

The gathering algorithm will be applied to the *ayah* (2:2). To gather the readings for an *ayah*, there is a need to know the different readings of its words (Table 1).

Table 1

Readings of the Words of the Ayah (2:2)

Rank	Word	Readings	Narrators	Reading Code
word1	*Thalika* (ذلك)	*Thalika* (ذلك)	All narrators	DEFAULT
word2	*Alkitab* (الكتاب)	*Alkitab* (الكتاب)	All narrators	DEFAULT
word3	La (لا)	*La* (لا^2)	All narrators	QASR_LA
		Laa (لا^4)	Khalaf, Khallad	MAD_LA
word4	*Rayba* (ريب)	*Rayba* (ريب)	All narrators	DEFAULT

(continued)

Rank	Word	Readings	Narrators	Reading Code
word5	*Feehi* (فيه)	*Feehe* (فيهِ) with *kasr alha*	All narrators except Bazzi and Qonbul	QASR_HA_ ALKINAYA
		Feehee ($^{2\sim}$فيهِ) with short *mad* with two *harakah* for *ha* (ـه)	Bazzi, Qonbul	SILAT_HA_ ALKINAYA
		Feehh (فيهّ) with *edgham* (ـه) into (هـ) in the next word	DoriAbuAmr, Soosi, Roways, Rawh	EDGHAM_ KABEER
word6	*Hudan* (هدىً)	*Hudal* (هُدالِّ) *edgham* without *ghonna*	All Narrators	EDGHAM_ WITHOUT_ GHONNA
		Hudall (^{2}هُدالِّ) *edgham* with *ghonna*	Qaloon, Warsh, Bazzi, Qonbul, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh	EDGHAM_ WITH_ GHONNA
word7	*Lilmuttaqeen* (للمتقين)	*Lilmuttaqeen* (للمتقين) without *ha sakt*	All Narrators	NO_HA_ SAKT
		Lilmuttaqeenah (للمتقينه) with *ha sakt*	Roways, Rawh	HA_SAKT

The *ayah* is (*Thalikal kitabu la rayba feehi hudal lilmuttaqeen* (ذَٰلِكَ الْكِتَابُ لَا رَيْبَ ۛ فِيهِ ۛ هُدًى لِلْمُتَّقِينَ ٢)*.* This *ayah* is composed of seven words. The second column shows the words in Arabic along with the English transliteration. The third column shows the different readings of the words. The fourth and fifth columns show the narrators and the *reading codes*. The reading code is used to refer to a letter reading. The reading of the word is identified by the reading code(s) for its letters. Letters of a word have a DEFAULT reading code except those with multiple readings. Not all words have multiple readings (e.g. *word-1*). The same narrator may have multiple readings for the same word (e.g. *Khalaf* and *Khallad* have two readings for the *word-3*). Narrators of different readings for the same word may overlap. A word may have more than two readings (e.g. *word-5* has three readings).

In addition to knowing the readings of the words of the *ayah*, contradicting readings for narrators should be known. If the *ayah* contains more than one word with multiple readings for the same narrator, some combinations of these words' readings may not be acceptable for that narrator and they should be avoided in reading the *ayah*. Table 2 shows the contradicting readings for the narrators for *ayah* (2:2).

Table 2

Contradicting Readings for Ayah (2:2)

Contradicting Reading	Contradicting Reading	Narrators
EDGHAM_KABEER	EDGHAM_WITHOUT_ GHONNA	Roways, Rawh
EDGHAM_KABEER	HA_SAKT	Roways, Rawh

For example, the second contradiction means that if a reciter reads this *ayah* for *Roways* and reads *word-5* with EDGHAM_KABEER reading, then he can only read *word-7* with NO_HA_SAKT reading. However, if he reads *word-5* without EDGHAM_KABEER for *Roways*, then *word-7* can be read with HA_SAKT or with NO_HA_SAKT. No general rules identify these contradictions. They should be learned and memorised by the reciter. The gathering of readings for this *ayah* is shown in Table 3. The reciter will go through nine passes in gathering the readings of this *ayah*. Each row in the table corresponds to one recitation pass through the *ayah*. The reciter reads the *ayah* as shown in the pass with the corresponding readings. The *ayah* does not need to be read completely for every narration.

Table 3

Gathering of Readings of Ayah (2:2)

Pass	Readings Gathering	Narrators	Reading Code
1	ذَٰلِكَ الْكِتَٰبُ لَا(1) رَيْبَ فِيهِ (2) هُدًى (3) لِلْمُتَّقِينَ (4)	All Narrators except Bazzi and Qonbul	(1) QASR_LA (2) QASR_HA_ ALKINAYA_PLUS_NO_ EDGHAM_KABEER (3) EDGHAM_WITHOUT_ GHUNNA (4) NO_HA_SAKT
2	لِلْمُتَّقِينَ (1)	Roways, Rawh	(1) HA_SAKT
3	هُدًى (1) لِلْمُتَّقِينَ (2)	Qaloon, Warsh, Dori-AbuAmr, Soosi, Hisham, Ibn-Thakwan, Hafs, Eisa, Ibn-Jammaz, Roways, Rawh	(1) EDGHAM_WITH_ GHUNNA (2) NO_HA_SAKT
4	لِلْمُتَّقِينَ (1)	Roways, Rawh	(1) HA_SAKT
5	فِيهِ (1) ۛ هُدًى (2) لِلْمُتَّقِينَ (3)	Bazzi, Qonbul	(1) SILAT_HA_ALKINAYA (2) EDGHAM_WITHOUT_ GHUNNA (3) NO_HA_SAKT
6	هُدًى (1) لِلْمُتَّقِينَ (2)	Bazzi, Qonbul	(1) EDGHAM_WITHOUT_ GHUNNA (2) NO_HA_SAKT
7	فِيهِ (1) ۛ هُدًى (2) لِلْمُتَّقِينَ (3)	Dori-AbuAmr, Soosi	(1) EDGHAM_KABEER (2) EDGHAM_WITHOUT_ GHUNNA (3) NO_HA_SAKT
8	هُدًى (1) لِلْمُتَّقِينَ (2)	Dori-AbuAmr, Soosi, Roways, Rawh	(1) EDGHAM_WITH_ GHUNNA (2) NO_HA_SAKT
9	لَا (1) رَيْبَ ۛ فِيهِ (2) هُدًى (3) لِلْمُتَّقِينَ (4)	Khalaf, Khallad	(1) MAD_LA (2) QASR_HA_ ALKINAYA_PLUS_NO_ EDGHAM_KABEER (3) EDGHAM_WITHOUT_ GHUNNA (4) NO_HA_SAKT

Table 4 shows the status by the end of the first pass. In this pass, the *ayah* is recited as shown in *Recited* row. Words that are read with one way only for all narrators have the DEFAULT reading code. *StoppingNarrators* are not presented in the form of pairs to save space. However, it is easy to generate these pairs from the table. For example, for *word-3, stoppingNarrators* set is *{(Khallaf,* MAD_LA*)*, *(Khallad,* MAD_LA*)}*. The first pass always covers the whole *ayah*. Other passes are usually shorter since they start from some word within the *ayah*; nevertheless, they always end at the end of the *ayah*.

Table 4

Status by the end of the Pass-1

Word	Recited	Stopping Narrators
1	DEFAULT	
2	DEFAULT	
3	QASR_LA	MAD_LA: Khalaf, Khallad
4	DEFAULT	
5	QASR_HA_ALKINAYA	SILAT_HA_ALKINAYA: Bazzi, Qonbul EDGHAM_KABEER: DoriAbuAmr, Soosi, Roways, Rawh
6	EDGHAM_WITHOUT_GHONNA	EDGHAM_WITH_GHONNA: Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh
7	NO_HA_SAKT	HA_SAKT: Roways, Rawh
Finished Partially: Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Khalaf, Khallad, Eisa, IbnJammaz, Roways, Rawh, **Finished Completely:** Shuba, AbuAlharith, DoriAlkisaee, Ishaq, Idrees		

Figure 2 traces the algorithm to show how the first pass is generated. Initially, the *currentWord* is *word-1* and *stoppingNarrators* set contains twenty pairs, one pair for each narrator with the DEFAULT reading. The *currentNarrator* will be *Qaloon* because he is the highest in rank among the narrators in *stoppingNarrators. CurrentNarrator* will not change until the end of the pass. All narrators will join *walkingNarrators* because all of them read *currentWord* like the *currentNarrator* (*Qaloon*). All the pairs, and hence narrators, will be removed from *stoppingNarrators* for *word-1* because none of them read it in any other way. Since *word-1* is not the last word in the *ayah*, *currentWord* becomes *word-2.* The *stoppingNarrators* at *word-2* will contain the twenty pairs: one pair for each narrator with the DEFAULT reading. A new iteration starts and the same will happen for *word-2* because it is read in one way only for all narrators. The *currentWord* becomes *word-3* with *stoppingNarrators* composed of twenty-two pairs: twenty pairs for the twenty narrators with the QASR_LA reading (which is the default reading for MAD_LA reading code). Another two pairs will be added: (*Khalaf*, MAD_LA) and (*Khallad*, MAD_LA).

The *currentWord* (*word-3*) is read with QASR_LA reading (*currentReading* will be QASR_LA). This is the only way with which *Qaloon* reads *word-3.* All narrators read *word-3* with *currentReading*, thus they will all join *walkingNarrators*. The *stoppingNarrators* for *word-3* will reduce down to {(*Khalaf*, MAD_LA), (*Khallad*, MAD_ LA)}. This is because they have another reading for *word-3* (MAD_ LA). Next, *currentWord* becomes *word-4* and its *stoppingNarrators* will be the twenty pairs for all narrators with DEFAULT reading code.

The *currentWord* (*word-4*) is read the same way for all narrators. All stopping narrators for *word-4* will join *walkingNarrators* set. The *stoppingNarrators* set for *word-4* will reduce down to an empty set. *CurrentWord* becomes *word-5*. *StoppingNarrators* for *word-5* will be {*(n,r)*| *n* is any narrator except *Bazzi* and *Qonbul* and *r* is QASR_HA_ ALKINAYA}∪{(*Bazzi*, SILAT_HA_ALKINAYA), (*Qonbul*, SILAT_ HA_ALKINAYA)} ∪ {(*DoriAbuAmr*, EDGHAM_KABEER), (*Soosi*, EDGHAM_KABEER),(*Roways*,EDGHAM_KABEER),(*Rawh*, EDGHAM_KABEER)}.

The *currentWord* (*word-5*) is read for *currentNarrator* (*Qaloon*) with *currentReading* of QASR-HA-ALKINAYA. Narrators that read with this reading will join the *walkingNarrators*. *WalkingNarrators* will include all narrators except *Bazzi* and *Qonbul*. These two narrators are excluded because they do not read *word-5* as the *currentReading*. All pairs *(n,r)* with *r* is QASR_HA_ALKINAYA will be removed from the *word-5.stoppingNarrators* of *word-5*. *StoppingNarrators* for *word-5* reduces to {(*Bazzi*, SILAT_HA_ALKINAYA), (*Qonbul*, SILAT_HA_ALKINAYA), (*DoriAbuAmr*, EDGHAM_KABEER), (*Soosi*, EDGHAM_KABEER), (*Roways*, EDGHAM_KABEER), (*Rawh*, EDGHAM_KABEER)}. Note that narrators with EDGHAM_KABEER will stay with the stopping narrators of *word-5*. These four narrators also read with QASR_HA_ALKINAYA reading, so their corresponding pairs will join with the walking narrators. *CurrentWord* becomes *word-6* with *stoppingNarrators* of {*(n,r)*| *n* is not *Bazzi* nor *Qonbul* and *r* is EDGHAM_WITHOUT_GHONNA)} ∪{*(x,*EDGHAM_WITH_GHONNA*)*| *x* is *Qaloon*, *Warsh*, *DoriAbuAmr*, *Soosi*, *Hisham*, *IbnThakwan*, *Hafs*, *Eisa*, *IbnJammaz*, *Roways* or *Rawh*}.

The *currentWord* (*word-6*) is read for *currentNarrator* (*Qaloon)* with *currentReading* is EDGHAM_WITHOUT_GHONNA. Note that both pairs {(*Qaloon*, EDGHAM_WITHOUT_GHONNA) and (*Qaloon*, EDGHAM_WITH_GHONNA)} are in *stoppingNarrators* set for *word-6*. The reciter will pick the first pair because the first reading code is the default of the second. *StoppingNarrators* at *word-6* are all narrators except *Bazzi* and *Qonbul* and all of them read with *currentReading*. Therefore, all of them will join *walkingNarrators*. Their corresponding pairs will be removed from the *stoppingNarrators* of *word-6*. *StoppingNarrators* for *word-6* will reduce to {(*x*,EDGHAM_WITH_GHONNA)| *x* is *Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways* or *Rawh*}. *CurrentWord* becomes *word-7.* The *stoppingNarrators* for *word-7* becomes {(*n*,NO_HA_SAKT)| *n* is not *Bazzi* nor *Qonbul*} ∪ {(*Roways*, HA_SAKT), (*Rawh*, HA_SAKT)}. Note that there are no contradictions between *currentReading* and any of the readings for the previous words. There are no readings yet with EDGHAM_KABEER (Table 2).

The *currentWord* (*word-7*) is read by *currentNarrator* (*Qaloon*) with *currentReading* (NO_HA_SAKT). All *stoppingNarrators* read with this reading, thus all of them will join the *walkingNarrators*. All pairs with the NO_HA_SAKT reading will be removed from the *stoppingNarrators* of *word-7*. Therefore, *stoppingNarrators* for *word-7* will reduce to the set {(*Roways*, HA_SAKT), (*Rawh*, HA_SAKT)}. Since *word-7* is the last word of the *ayah*, *currentWord* is set to be the word - with stopping narrators - that is closest to the end of the *ayah*, which is *word-7* itself. As a result, *currentWord* becomes *word-7*. By this, the first pass is finished and the situation will be as shown in Table 4. Partially finished narrators are those who are still stopping at some word of the *ayah*. They still have some other readings to come in later passes. The completely finished narrators are not stopping anywhere and all their readings are covered.

The second pass will start with the right backward arrow in Figure 2. The *currentWord* is *word-7* with *stoppingNarrators* of {(*Roways*, HA_SAKT), (*Rawh*, HA_SAKT)}. The *currentNarrator* is set to be the narrator with the highest rank among the *stoppingNarrators* of *word-7* (that is, *Roways*). The reciter will read *word-7* with (HA_SAKT) for *currentNarrator* (*Roways)*. Both *Roways* and *Rawh* join *WalkingNarrators*. Both narrators will also be removed from *stoppingNarrators* for *word-7*. Therefore, *stoppingNarrators* for *word-7* becomes empty. Since *word-7* is the last word in the *ayah*, *currentWord* will be set to the word with stopping narrators that are closest to the end of the *ayah*. Thus, it will be set to *word-6* and a new pass starts.

In pass-2, the reciter does not start reading from the beginning of the *ayah* because this earlier part of the *ayah* has been covered implicitly in the nearest previous pass(es). Although one word only is read in this pass, this pass corresponds to the narration for the whole *ayah*. The narrations of the whole *ayah* can be rebuilt by concatenating the part that is read in this pass, with the latest readings of the preceding words in the nearest previous pass(es). These last readings are stored in *wordReadingVector.* For pass-2, the corresponding whole *ayah* narration is the same as the first pass, except that *word-7* is to be read with HA_SAKT reading. By the end of this pass, *Roways* and *Rawh* have finished partially because they are still stopping at *word-5* and *word-6*.

The following section will explore pass-7 to illustrate the case where contradicting readings for a narrator co-exist. The stopping narrators at the end of pass-6 are shown in Table 5. *CurrentWord* is set to be the closest word to the end of the *ayah* with stopping narrators (*word-5*). The *stoppingNarrators* set for *word-5* is {(*DoriAbuAmr*, EDGHAM_KABEER), (*Soosi*, EDGHAM_KABEER), (*Roways*, EDGHAM_KABEER), (*Rawh*, EDGHAM_KABEER)}.*CurrentNarrator* will be *DoriAbuAmr* (highest in rank among *stoppingNarrators).* The reciter will read *word-5* for *currentNarrator* with EDGHAM_KABEER. All stopping narrators read like *currentNarrator*, thus four of them will join the *walkingNarrator* and leave *stoppingNarrators* for *word-5*. *CurrentWord* becomes *word-6* with stopping narrators is {(*DoriAbuAmr*, EDGHAM_WITHOUT_GHONNA), (*DoriAbuAmr*, EDGHAM_WITH_GHONNA), (*Soosi*, EDGHAM_WITHOUT_GHONNA), (*Soosi*, EDGHAM_WITH_GHONNA), (*Roways*, EDGHAM_WITH_GHONNA), (*Rawh*,EDGHAM_WITH_GHONNA)}. Note that (*Roways*, EDGHAM_WITHOUT_GHONNA) and (*Rawh*, EDGHAM_WITHOUT_GHONNA) are not added to the *stoppingNarrators* for *word-6* because there is a contradiction between EDGHAM_WITHOUT_GHONNA and EDGHAM_KABEER for these two narrators (Table 2).

Table 5

Status by the end of Pass-6

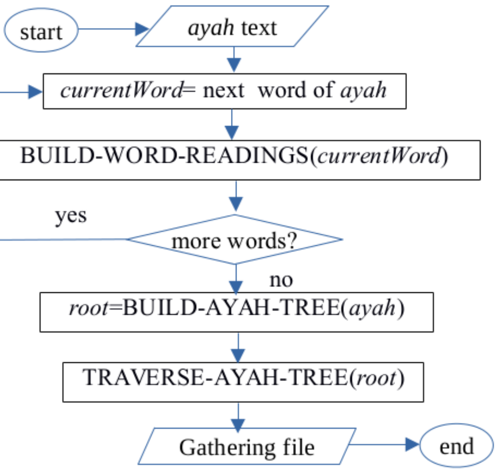
Word	1	2	3	4	5	6	7
Stopping narrators			(لا4) MAD_LA Khalaf, Khallad		*Feehh* (فيهّ) EDGHAM_KABEER DoriAbuAmr, Soosi, Roways,Rawh		

AUTOMATION OF THE GATHERING PROCESS

A software system was developed to automate the process of gathering of Quranic readings. The main algorithm (Figure 3) takes the text of the *ayah* as input to gather readings. The programme goes through the *ayah* word by word by summoning BUILD-WORD-READINGS to identify the different readings for the words. Then, the *ayah* readings tree is built and traversed to generate the gathering text file.

Figure 3

GATHER-READINGS Algorithm


start
ayah text
currentWord= next word of *ayah*
BUILD-WORD-READINGS(*currentWord*)
yes
more words?
no
root=BUILD-AYAH-TREE(*ayah*)
TRAVERSE-AYAH-TREE(*root*)
Gathering file
end

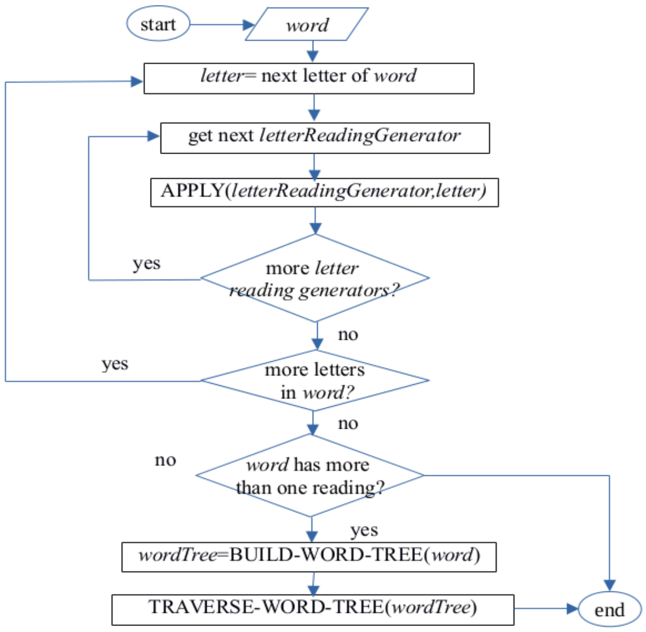
Building Word Readings Tree

The BUILD-WORD-READINGS algorithm (Figure 4) takes a *word* as input. The *letter Reading Generators* are applied one after another on the letters of *word* via the APPLY algorithm. A letter reading generator is the software unit that detects and defines a reading for a letter. It is identified by a *condition* and a *letter reading code*. If one letter of *word* has more than one reading, then *word* has multiple readings. Once the readings of all the letters are determined, the readings of *word* as a whole are constructed by summoning BUILD-WORD-TREE and TRAVERSE-WORD-TREE algorithms.

The *APPLY* algorithm (Figure 5) takes *letterReadingGenerator* and *letter* as input. If *letter* satisfies the *letterReadingGenerator*'s condition, a *letterReading* object is created by identifying the corresponding *letter reading code* and is then added to *letter*'s readings list. Before applying letter reading generators, every letter has only one letter reading object in its readings list (called *firstReading* with DEFAULT *reading code*). When a new *letterReading* is added to the letter's reading list, the *firstReading* code should be updated. If it is the DEFAULT reading, the *firstReading* code is set to be the default reading code of *letterReading*. Otherwise, it is set to be the composition of the *firstReading* code and the default reading code of *letterReading*.

Figure 4

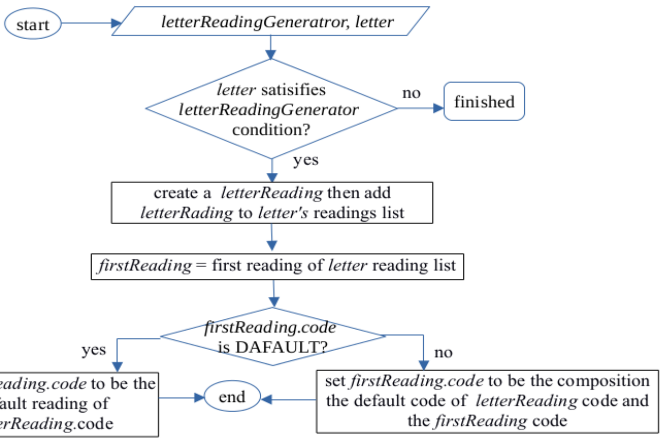
BUILD-WORD-READINGS Algorithm


start
word
letter= next letter of word
get next letterReadingGenerator
APPLY(letterReadingGenerator,letter)
yes
more letter reading generators?
no
yes
more letters in word?
no
no
word has more than one reading?
yes
wordTree=BUILD-WORD-TREE(word)
TRAVERSE-WORD-TREE(wordTree)
end

To illustrate the APPLY algorithm, take for example *word-5* (فيه) in the *ayah* example. Two letter readings generator conditions are satisfied at the letter (ـه): *silat_ha_alkinaya_generator*, and *edgham_kabeer_generator*. First, a *letter Reading* object - with SILAT_HA_ALKINAYA reading code - is created and added to the readings list of the letter (ـه). The letter reading codes attributes are maintained in a database. These attributes include *description*, *contradicting reading codes*, *default reading code*, and *narrators*. Every reading code has a *default* reading code that is associated with it. The default of a reading code can be viewed as its opposite. The default reading code for SILAT_HA_ALKINAYA is QASR_HA_ALKINAYA. Adding SILAT_HA_ALKINAYA letter reading to the letters reading list of the letter (ـه) , will change the *firstReading* code from DEFAULT to QASR_HA_ALKINAYA (Figure 6).

Figure 5

APPLY Algorithm


start
letterReadingGeneratror, letter
letter satisifies letterReadingGenerator condition?
no
finished
yes
create a letterReading then add letterRading to letter's readings list
firstReading = first reading of letter reading list
firstReading.code is DAFAULT?
yes
no
set firstReading.code to be the default reading of letterReading.code
end
set firstReading.code to be the composition the default code of letterReading code and the firstReading code

When *edgham_kabeer_generator* is applied, another *letterReading* will be created and added to the readings list of the letter (ـهـ). Since the *firstReading* code is not DEFAULT, its code will be changed to be the default composite of the two letter readings that were added to the letter (Figure 7).

Figure 6

The Letter Readings List for the Letter (ـهـ) After Applying Silat_ha_alkinaya_generator

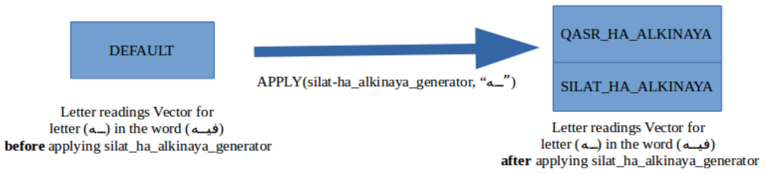
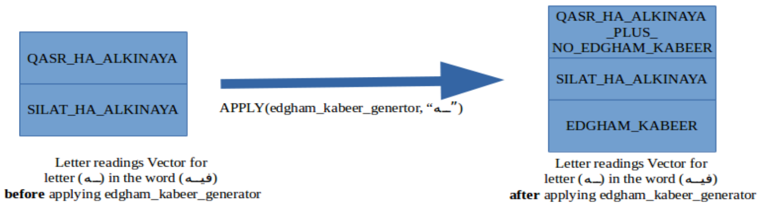

DEFAULT
APPLY(silat-ha_alkinaya_generator, "ـهـ")
QASR_HA_ALKINAYA
SILAT_HA_ALKINAYA
Letter readings Vector for letter (ـهـ) in the word (فيه) before applying silat_ha_alkinaya_generator
Letter readings Vector for letter (ـهـ) in the word (فيه) after applying silat_ha_alkinaya_generator

Figure 7

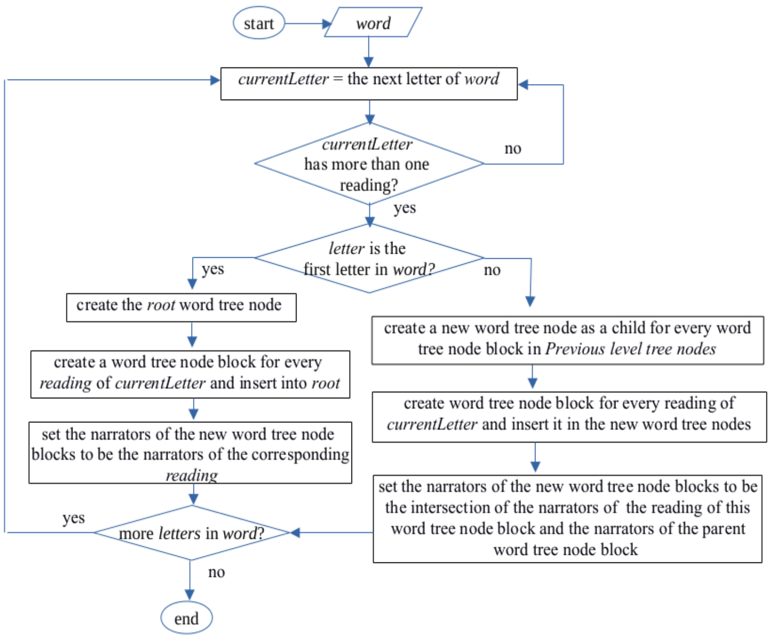
The Letter Readings for the Letter (ـهـ) After Applying Edgham_kabeer_generator


QASR_HA_ALKINAYA
SILAT_HA_ALKINAYA
APPLY(edgham_kabeer_genertor, "ـهـ")
QASR_HA_ALKINAYA _PLUS_ NO_EDGHAM_KABEER
SILAT_HA_ALKINAYA
EDGHAM_KABEER
Letter readings Vector for letter (ـهـ) in the word (فيه) before applying edgham_kabeer_generator
Letter readings Vector for letter (ـهـ) in the word (فيه) after applying edgham_kabeer_generator

BUILD-WORD-TREE algorithm (Figure 8) takes a *word* with one or more letters with multiple readings as input. The *word* letters are taken one by one. If *currentLetter* is the first letter in *word* with multiple readings, the *root* word tree node is created. A *tree node block* will be created for every reading of *currentWord* and is then inserted into the *root*. The narrators of the *root* tree node blocks will be set as the narrators of the corresponding letter reading. Every *currentLetter* in *word* with multiple readings will result in adding a new level to the word tree. A new level is created by creating new *word tree nodes* to be the children of every *word tree node block* in the previous level. For every reading of the *currentLetter*, a new word tree node block is created and inserted into the new level tree nodes. Narrators for the new *word tree node blocks* will be set as the *intersection* of the narrators of the newly created word tree node blocks and the narrators of the parent word tree node block. If this intersection is empty, then the corresponding tree node block is eliminated from its tree node.

Figure 8

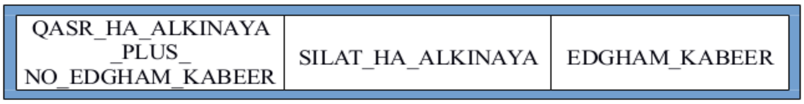
BUILD-WORD-TREE Algorithm


start
word
currentLetter = the next letter of word
currentLetter has more than one reading?
no
yes
letter is the first letter in word?
yes
no
create the root word tree node
create a word tree node block for every reading of currentLetter and insert into root
set the narrators of the new word tree node blocks to be the narrators of the corresponding reading
create a new word tree node as a child for every word tree node block in Previous level tree nodes
create word tree node block for every reading of currentLetter and insert it in the new word tree nodes
set the narrators of the new word tree node blocks to be the intersection of the narrators of the reading of this word tree node block and the narrators of the parent word tree node block
yes
more letters in word?
no
end

The word tree for *word-5* is shown in Figure 9. The word tree is composed of one tree node only (the *root*). This is because there is only one letter in this word with multiple readings. The word tree node of *word-5* consists of three *tree node blocks* because there are three different readings for(ـه).

Figure 9

The Word Tree Node for Word-5 with Three Tree Node Blocks



QASR_HA_ALKINAYA _PLUS_ NO_EDGHAM_KABEER	SILAT_HA_ALKINAYA	EDGHAM_KABEER

The word tree for the *word-5* example is too simple. To better illustrate the algorithm, a tree will be built for a word with many letters and multiple readings. Consider the word *Jibreel* (جِبْرِيل)with letter readings shown in Table 6. There are two different readings for the letter(جـ), two different readings for the letter (ـر), and three different readings for the letter (ـيـ).

Table 6

Letter Readings of the Word Jibreel(جبريل)

Letter	Readings	Letter Reading Code	Narrators
جـ	(ja) جَـ	FATH_JEEM_JIBREEL	Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh
	(ji) جِـ	KASR_JEEM_JIBREEL	Bazzi, Qonbul, Shuba, Khalaf, Khallad, AbuAlharith, DoriAlkisaee, Ishaq, Edrees
ـبـ	ـبْـ	DEFAULT	All narrators
ـر	(ri) ـرِ	KASR_RA_JIBREEL	Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh, Bazzi, Qonbul
	(ra) ـرَ	FATH_RA_JIBREEL	Shuba, khalaf, khallad, AbuAlharith, DoriAlkisaee, Ishaq, Edrees

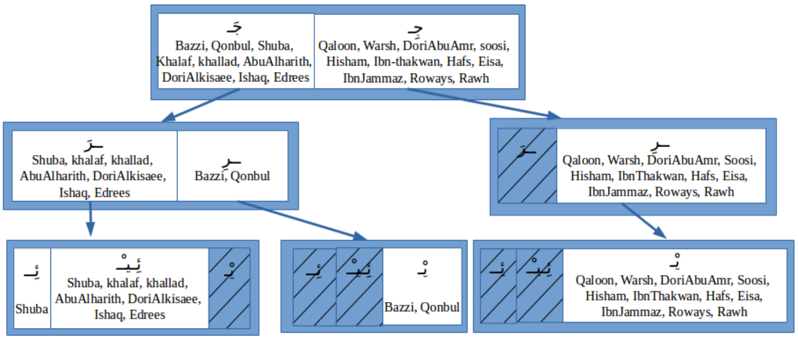
(continued)

Letter	Readings	Letter Reading Code	Narrators
يـ	يْـ	ISKAN_YA_JIBREEL	Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh, Bazzi, Qonbul
	ئِيْـ	ISKAN_YA_JIBREEL_WITH_HAMZ_BEFORE	Bazzi, Qonbul, Shuba, khalaf, khallad, AbuAlharith, DoriAlkisaee, Ishaq, Edrees
ـل	ئِـ	HATHF_YA_JIBREEL_WITH_HAMZ	Shuba
	ـل	DEFAULT	All narrators

The word tree will be built as follows: first, the root is built, which corresponds to the letter readings of the first letter in the word with multiple readings (ـجـ) (Figure 10). In the second level, a word tree node is added for every word tree node block in the level above. A new word tree node block will be added for every reading for the letter (ـر) in the newly created tree nodes. The set of narrators of a new word tree node block will be the narrators' intersection of the letter reading with the narrators in the parent word tree node block. For example, the narrators in the word tree node block of (ـرِ) in the left word tree node are *Bazzi* and *Qonbul*. This was found by intersecting the narrators in the parent word tree node block (ـجَـ) with the narrators of the (ـرِ) reading. Tree node blocks with no narrators are shaded because there are no narrators in the intersection. These word tree node blocks should be eliminated; however, they are kept in the figure for illustration only. The same process is repeated for the third letter with multiple readings (ـيـ).

Figure 10

The Word Tree for the Word (Jibreel)


جَ
Bazzi, Qonbul, Shuba, Khalaf, khallad, AbuAlharith, DoriAlkisaee, Ishaq, Edrees
جِ
Qaloon, Warsh, DoriAbuAmr, soosi, Hisham, Ibn-thakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh
رَ
Shuba, khalaf, khallad, AbuAlharith, DoriAlkisaee, Ishaq, Edrees
رِ
Bazzi, Qonbul
رِ
Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh
ئِ
Shuba
ئِيـ
Shuba, khalaf, khallad, AbuAlharith, DoriAlkisaee, Ishaq, Edrees
يْ
Bazzi, Qonbul
يْ
Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh

Every path from a root word tree node block to a non-shaded *leaf* word tree node block corresponds to one reading for the word as a whole. The narrators of this word reading are the narrators of the corresponding leaf tree node block (Table 7). The reading code of the word is the set of the reading codes of its letters.

Table 7

Word Readings for the Word(جبريل)*Extracted from Word Tree*

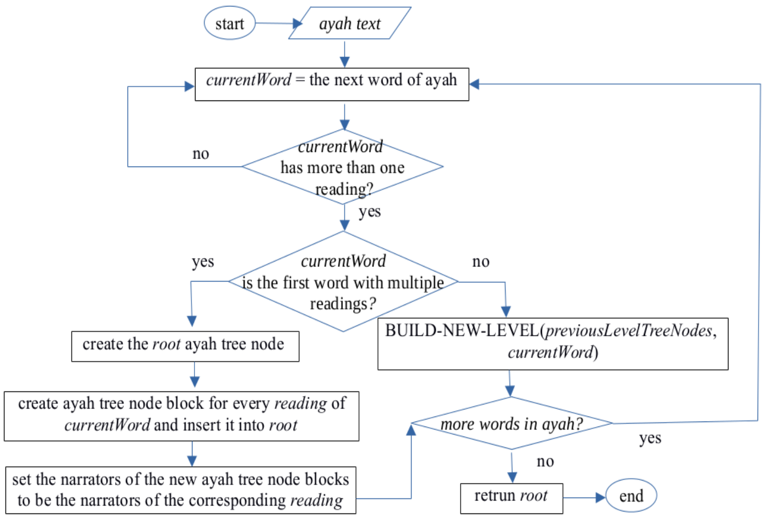
Readings	Narrators	Word Reading Code
جِبْرِيل *Jibreel*	Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh	KASR_JEEM_JIBREEL KASR_RA_JIBREEL ISKAN_YA_JIBREEL
جَبْرِيل *Jabreel*	Bazzi, Qonbul	FATH_JEEM_JIBREEL KASR_RA_JIBREEL ISKAN_YA_JIBREEL
جَبْرَئِيل *Jabra'eel*	Shuba, khalaf, khallad, AbuAlharith,DoriAlkisaee, Ishaq, Edrees	FATH_JEEM_JIBREEL FATH_RA_JIBREEL ISKAN_YA_JIBREEL_ WITH_HAMZ_BEFORE
جَبْرَئِل *Jabra'el*	Shuba	FATH_JEEM_JIBREEL FATH_RA_JIBREEL HATHF_YA_JIBREEL_ WITH_HAMZ

Ayah Readings Tree

Once the different readings of the words of the *ayah* are defined, the different readings of the *ayah* as a whole can be defined by building *ayah readings tree*. The BUILD-AYAH-TREE algorithm (Figure 11) is beckoned by the main algorithm (Figure 3). Building the *ayah* reading tree is very similar to building the word reading tree, except that it runs on the word readings instead of letter readings. The algorithm goes through the words of the *ayah* word by word. If *currentWord* has only one reading, it becomes the next word of the *ayah*. If *currentWord* is the first word with multiple readings in the *ayah*, the algorithm will build the *root* tree node with a tree node block for every reading of *currentWord.* Narrators are set to be the narrators of the corresponding reading. Then, *currentWord* will be set to be the next word of the *ayah*. If the *currentWord* is not the first word with multiple readings in the *ayah*, a new level is built into the tree by summoning BUILD-NEW-LEVEL (Figure 12). Finally, the *root* of the *ayah* readings tree will be returned after visiting all the words of the *ayah*. If the *ayah* has no words with multiple readings, no tree will be built. To be precise, there is only one reading for this *ayah*.

Figure 11

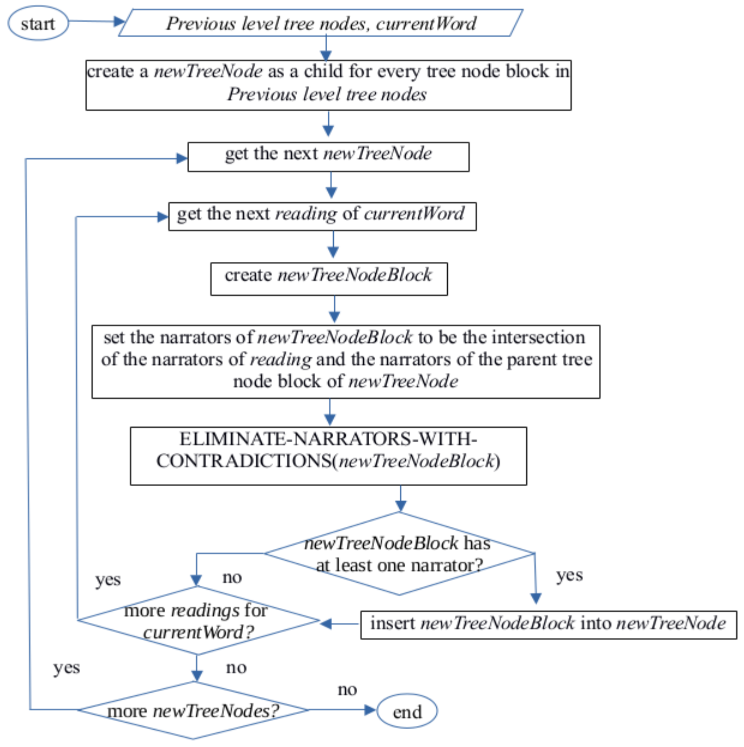
BUILD-AYAH-TREE Algorithm


start
ayah text
currentWord = the next word of ayah
currentWord has more than one reading?
no
yes
currentWord is the first word with multiple readings?
yes
no
create the root ayah tree node
BUILD-NEW-LEVEL(previousLevelTreeNodes, currentWord)
create ayah tree node block for every reading of currentWord and insert it into root
more words in ayah?
yes
no
set the narrators of the new ayah tree node blocks to be the narrators of the corresponding reading
retrun root
end

The BUILD-NEW-LEVEL algorithm (Figure 12) is beckoned in case the *currentWord* is not the first word with multiple readings. A new *ayah* tree node will be created as a child for every *ayah tree node block* in the previous level. An *ayah tree node block* will be created for every *reading* for *currentWord* with narrators to be the intersection of corresponding *reading* and the narrators of the parent tree node block. Narrators of the new tree node block with contradictions and an ancestor tree node block reading will be eliminated by summoning the ELIMINATE-NARRATORS-WITH-CONTRADICTIONS algorithm. Only if the narrators set of the new *tree node block* is not empty, it will be inserted into the new *ayah tree node*.

Figure 12

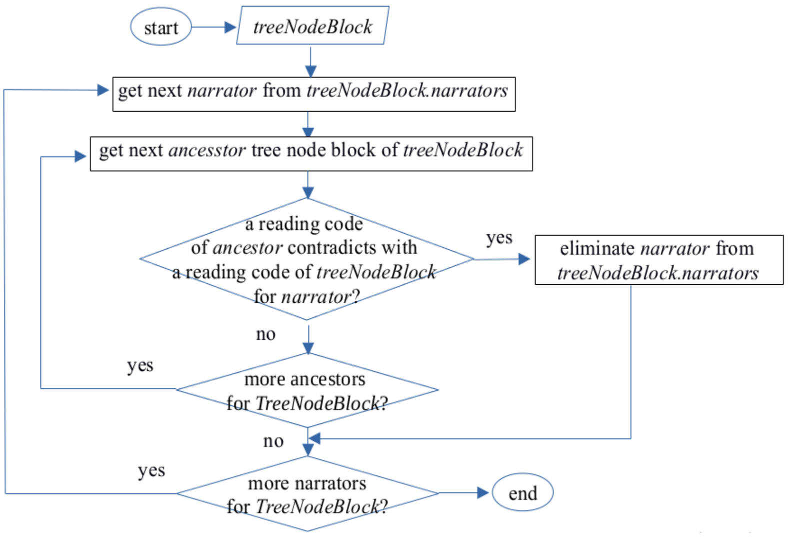
BUILD-NEW-LEVEL Algorithm


start
Previous level tree nodes, currentWord
create a newTreeNode as a child for every tree node block in Previous level tree nodes
get the next newTreeNode
get the next reading of currentWord
create newTreeNodeBlock
set the narrators of newTreeNodeBlock to be the intersection of the narrators of reading and the narrators of the parent tree node block of newTreeNode
ELIMINATE-NARRATORS-WITH-CONTRADICTIONS(newTreeNodeBlock)
newTreeNodeBlock has at least one narrator?
yes
no
insert newTreeNodeBlock into newTreeNode
more readings for currentWord?
yes
no
more newTreeNodes?
yes
no
end

The ELIMINATE-NARRATORS-WITH-CONTRADICTIONS algorithm (Figure 13) is beckoned before adding a new *ayah* tree node block into a new *ayah* tree node for *currentWord* (in *ayah* readings tree). Its input is the newly created *treeNodeBloc*k. The narrators of the new *treeNodeBlock* are taken one by one. It checks if a contradiction exists between one of the readings of the new *treeNodeBlock* with one of the readings of an ancestor tree node block for *narrator*. If a contradiction is found, that *narrator* is eliminated.

Figure 13

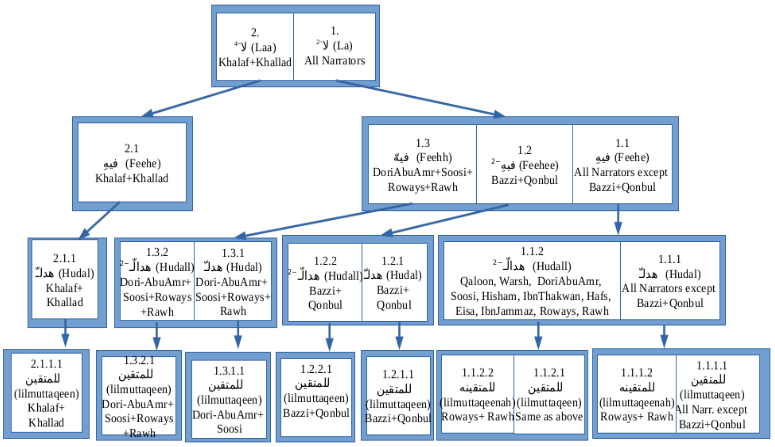
ELIMINATE-NARRATORS-WITH CONTRADICTIONS Algorithm


start
treeNodeBlock
get next narrator from treeNodeBlock.narrators
get next ancesstor tree node block of treeNodeBlock
a reading code of ancestor contradicts with a reading code of treeNodeBlock for narrator?
yes
eliminate narrator from treeNodeBlock.narrators
no
yes
more ancestors for TreeNodeBlock?
no
yes
more narrators for TreeNodeBlock?
end

The *ayah* readings tree for the example *ayah* (2:2) is shown in Figure 14. Every tree node block is identified by a sequence of numbers separated by dots. Each number represents the rank of the ancestor tree node block in its tree node. For example, the tree node block (1.2.2.1) is the tree node block that exists in the fourth level and descends from tree node block (1.2.2) in the third level, which descends from tree node block (1.2) in the second level, which descends from tree node block (1.) in the root. The tree node and tree node blocks within a tree node are ranked from right to left.

Figure 14

Ayah Readings Tree


2. لا~4 (Laa) Khalaf+Khallad
1. لا~2 (La) All Narrators
2.1 فيهِ (Feehe) Khalaf+Khallad
1.3 فيهْ (Feehh) DoriAbuAmr+Soosi+ Roways+Rawh
1.2 فيهِ~2 (Feehee) Bazzi+Qonbul
1.1 فيهِ (Feehe) All Narrators except Bazzi+Qonbul
2.1.1 هدًى (Hudal) Khalaf+ Khallad
1.3.2 هدالّ~2 (Hudall) Dori-AbuAmr+ Soosi+Roways +Rawh
1.3.1 هدًى (Hudal) Dori-AbuAmr+ Soosi+Roways+ Rawh
1.2.2 هدالّ~2 (Hudall) Bazzi+ Qonbul
1.2.1 هدًى (Hudal) Bazzi+ Qonbul
1.1.2 هدالّ~2 (Hudall) Qaloon, Warsh, DoriAbuAmr, Soosi, Hisham, IbnThakwan, Hafs, Eisa, IbnJammaz, Roways, Rawh
1.1.1 هدًى (Hudal) All Narrators except Bazzi+Qonbul
2.1.1.1 للمتقين (lilmuttaqeen) Khalaf+ Khallad
1.3.2.1 للمتقين (lilmuttaqeen) Dori-AbuAmr+ Soosi+Roways +Rawh
1.3.1.1 للمتقين (lilmuttaqeen) Dori-AbuAmr+ Soosi
1.2.2.1 للمتقين (lilmuttaqeen) Bazzi+Qonbul
1.2.1.1 للمتقين (lilmuttaqeen) Bazzi+Qonbul
1.1.2.2 للمتقينه (lilmuttaqeenah) Roways+ Rawh
1.1.2.1 للمتقين (lilmuttaqeen) Same as above
1.1.1.2 للمتقينه (lilmuttaqeenah) Roways+ Rawh
1.1.1.1 للمتقين (lilmuttaqeen) All Narr. except Bazzi+Qonbul

Generation of the *Ayah* Readings Gathering

After building the *ayah* readings tree, the tree is traversed in a well-defined order using the TRAVERSE-AYAH-TREE (*root*) algorithm to generate readings gathering text file (Figure 15). First, the *nextReading* of a given *currentReading* has to be defined. R_{next} is beckoned to be the next reading of $R_{current}$ if R_{next} satisfies one of the following four conditions. Remember that tree node blocks within tree nodes represent readings. Note that the tree nodes and readings within tree nodes are ranked from right to left (to match Arabic text direction). The first tree node in its level is the rightmost tree node, and the first reading is the rightmost reading in its tree node.

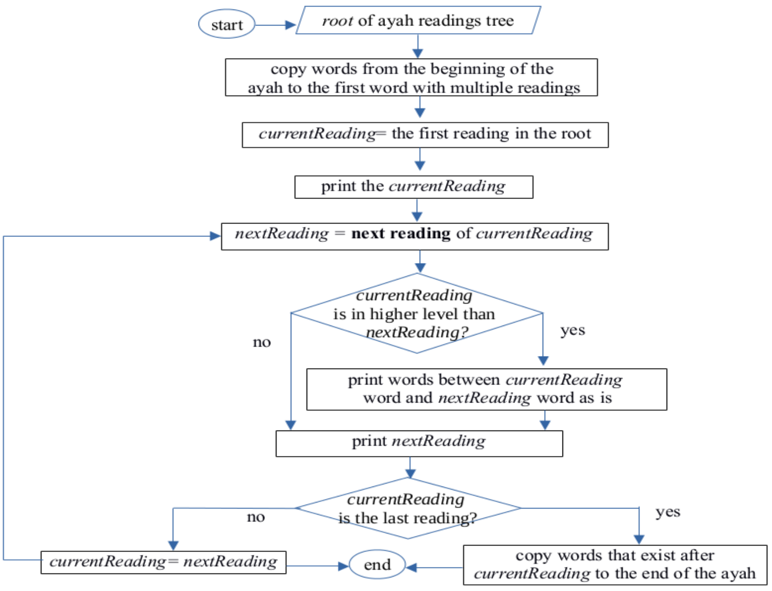
1. If $R_{current}$ has a child, R_{next} is the first reading in the tree node that is the child of $R_{current}$. For example, in Figure 14, if $R_{current}$ is the reading (1.), then R_{next} is the reading (1.1).
2. If $R_{current}$ exists in the last level of the *ayah* readings tree and is not the last reading in its node, then R_{next} will be the reading that exists immediately next to $R_{current}$. For example, in Figure 14, if $R_{current}$ is the reading (1.1.1.1), then R_{next} is the reading (1.1.1.2).
3. If $R_{current}$ exists in the last level of the *ayah* readings tree, is the last reading in its node, and its parent reading is not the last

reading in its node, then R_{next} is the reading that exists directly next to its parent reading. For example, in Figure 14, if $R_{current}$ is the reading (1.1.1.2), then R_{next} is the reading (1.1.2).

4. If $R_{current}$ exists in the last level of the *ayah* readings tree, is the last reading in its node, and its parent reading is also the last reading in its node, R_{next} will be the reading that exists immediately next to the closest ancestor of $R_{current}$ that is not the last in its node. For example, in Figure 14, if $R_{current}$ is the reading (1.2.2.1), then R_{next} is the reading (1.3) and if $R_{current}$ is the reading (1.3.2.1), then R_{next} is the reading (2.)

Figure 15

TRAVERSE-AYAH-TREE Algorithm


start
root of ayah readings tree
copy words from the beginning of the ayah to the first word with multiple readings
currentReading= the first reading in the root
print the currentReading
nextReading = next reading of currentReading
currentReading is in higher level than nextReading?
no
yes
print words between currentReading word and nextReading word as is
print nextReading
currentReading is the last reading?
no
yes
currentReading= nextReading
end
copy words that exist after currentReading to the end of the ayah

Summoning TRAVERSE-AYAH-TREE for the root of the tree in Figure 14 will result in visiting the tree node blocks in the following order: (1.), (1.1), (1.1.1), then (1.1.1.1) that correspond to pass-1; (1.1.1.2) that corresponds to pass-2; (1.1.2) then (1.1.2.1) that correspond to pass-3; (1.1.2.2) that corresponds to pass-4; (1.2), (1.2.1), then (1.2.1.1) that correspond to pass-5; (1.2.2) then (1.2.2.1) that correspond to pass-6; (1.3), (1.3.1), then (1.3.1.1) that correspond to pass-7; (1.3.2) then (1.3.2.1) that correspond to pass-8; (2.), (2.1), (2.1.1), then (2.1.1.1) that correspond to pass-9. The contents of the gathering file are summarised for *ayah* (2:2) in Table 3.

IMPLEMENTATION ISSUES

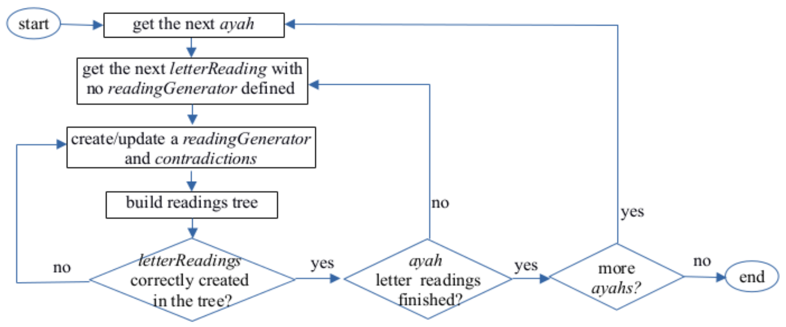
The present study implemented this system using Java following the object-orientation paradigm in analysis, design, and implementation. So far, the gathering was performed for Surah Al-Baqarah, which contains 286 *ayahs* with varying lengths.

System Validation

To assure the correctness of the generated gathering, the system was developed through a test-driven incremental process (Figure 16). First, the AYAH_TRAVERSE_TREE algorithm - that represents the gathering generation engine - was carefully programmed and tested. By this, it is guaranteed that any mismatch between the gathering that is generated by the system and the gathering found in the specialised gathering books occurs due to an incorrect *ayah* readings tree. An incorrect creation of *ayah* readings tree may result from: (1) a missing letter reading that was not added to a letter; or (2) wrongly adding a letter reading that should not be added; or (3) missing contradiction that should be defined. A missing letter reading may happen because the corresponding letter reading generator is not yet defined or due to a fault in its programming. The *ayahs* of Surah Al-Baqarah were taken one by one. This research referred to specialised books that described the readings of the words of the *ayahs* (Abdeen, 2006; Ibnu Al-Jazari, 2003; Salem, 2003; Tulba & Awadh, 2016). A word reading is viewed as a set of letter readings. For every new class of a letter reading, a *reading generator* was created by defining its *condition* and the corresponding *reading code*. Then, the *ayah* readings tree was built and tested. If the new letter reading was correctly modelled in the tree, the next step was to proceed to the next letter reading in the *ayah* that was not yet programmed. The process was repeated for every letter reading until all the *ayah* readings - and contradictions if any - were correctly and completely defined. If a mismatch was found, the reading generators were updated and new reading contradictions were formulated and stored. Then, the *ayah* readings tree was tested again. Once all the readings and contradictions for the current *ayah* were correct and complete, the next *ayah* was proceeded and the process was repeated.

Figure 16

System Development Process


start
get the next *ayah*
get the next *letterReading* with no *readingGenerator* defined
create/update a *readingGenerator* and *contradictions*
build readings tree
letterReadings correctly created in the tree?
no
yes
ayah letter readings finished?
no
yes
more *ayahs?*
yes
no
end

Modelling the Holy Quran

The Holy Quran was modelled using the *composite design pattern*. The Quran is modelled as a vector of *Surahs*, a *surah* is modelled as a vector of *ayahs*, an *ayah* is modelled as a vector of words, and a *word* is modelled as a vector of *letters*. This research used three Quranic text files from (Tanzil Documents, n. d.): *Uthmani, Simple*, and *Clean*. Every Quranic word was maintained in three forms: *Uthmani* where the word text was as written in the *Mos-haf*, *simple* that used standard Arabic dictation with annotation signs (*harakat*), and finally *clean* that used standard Arabic dictation but without annotation signs. These forms were necessary for the Quranic word to ease writing reading generators' conditions.

Reading Generators

A *reading generator* is the component that defines the letter readings. Every reading generator is implemented as a Java class that contains all the functionality needed to define the readings and associate them with the Quranic letters. A reading generator is identified by a *condition* and a *reading code*, another functionality that is done by the generator that is identical for all generators. Reading code is modelled as a Java class with the attributes that are needed to define the reading. There are more than 300 generators and this makes the manual definition of the generators hard. Therefore, the present study defined

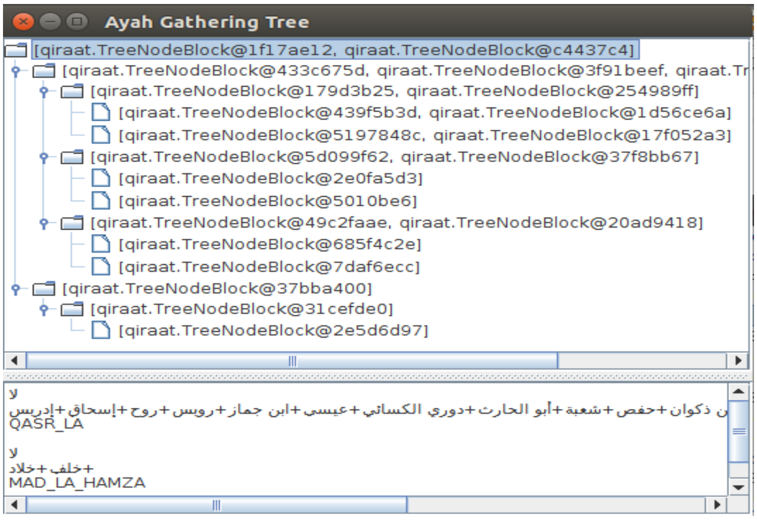
LetterReadingGeneratorGenertor that automatically generated the reading generator code file given the *condition* and the *reading code*. Readings generators were instantiated into a vector and applied one by one on each of the letters of the input *ayah*.

Graphic Representation of the *Ayah* Readings Tree

A simple Graphical User Interface (GUI) to view the *ayah* readings tree using JTree was developed (Elliott et al., 2002). This GUI was very practical in verifying the letter generators and the *ayah* readings tree during system development. The *ayah* tree example in Figure 14 is shown in Figure 17. It can be seen that every tree node in the *ayah* gathering tree is represented by a JTree node. There are seven leaf tree nodes with nine leaf tree node blocks that correspond to the nine passes. There are two leaf tree nodes with two tree node blocks each. The selected JTree node contents are shown in the lower panel.

Figure 17

Ayah Gathering Tree in the Form of Jtree


Ayah Gathering Tree

```
[qiraat.TreeNodeBlock@1f17ae12, qiraat.TreeNodeBlock@c4437c4]
  [qiraat.TreeNodeBlock@433c675d, qiraat.TreeNodeBlock@3f91beef, qiraat.Tr
    [qiraat.TreeNodeBlock@179d3b25, qiraat.TreeNodeBlock@254989ff]
      [qiraat.TreeNodeBlock@439f5b3d, qiraat.TreeNodeBlock@1d56ce6a]
      [qiraat.TreeNodeBlock@5197848c, qiraat.TreeNodeBlock@17f052a3]
    [qiraat.TreeNodeBlock@5d099f62, qiraat.TreeNodeBlock@37f8bb67]
      [qiraat.TreeNodeBlock@2e0fa5d3]
      [qiraat.TreeNodeBlock@5010be6]
    [qiraat.TreeNodeBlock@49c2faae, qiraat.TreeNodeBlock@20ad9418]
      [qiraat.TreeNodeBlock@685f4c2e]
      [qiraat.TreeNodeBlock@7daf6ecc]
  [qiraat.TreeNodeBlock@37bba400]
    [qiraat.TreeNodeBlock@31cefde0]
      [qiraat.TreeNodeBlock@2e5d6d97]
```

```
لا
ن ذكوان+حفص+شعبة+أبو الحارث+دوري الكسائي+عيسى+ابن جماز+رويس+روح+إسحاق+إدريس
QASR_LA

لا
+خلف+خلاد
MAD_LA_HAMZA
```

CONCLUSION

In this paper, the gathering of the Quranic readings process was formalised and implemented by a software system. It is believed that this work is useful because it proved that a detailed gathering of Quranic readings can be generated using computers, which would be very hard to do manually. Moreover, this software system would be a good base to build upon more software systems to serve the field of Quranic readings like vocal generation of Quranic readings gathering. As future work, it is recommended to complete the system to cover the whole Quran and make it available for interested students. It would be interesting to generalise the gathering process to work for any number of narrations that are decided by the user.

ACKNOWLEDGMENT

The authors gratefully acknowledge the use of the facilities of Al-Ahliyya Amman University. This research received no specific grants from any funding agency in the public, commercial, or non-profit sectors.

REFERENCES

Abdeen, A. (2016). *Al-Konoos Althameena fee Jam'e alqiraat alashr min tareeq tayyabat Annashr Alama Jaa'a fee tanqeeh alkareem.* (Alkonooz-Athameena in gathering the readings from Tayyibat Annashr ways for what came in Tanqeeh Alkareem). https://archive.org/details/sa71mir_gmail_20160411

Abdou, S., & Rashwan, M. (2014). A computer aided pronunciation learning system for teaching the Holy Quran recitation rules. In *IEEE/ACS 11th International Conference on Computer Systems and Applications* (pp. 543–550). https://doi.org/10.1109/aiccsa.2014.7073246

Abdullah, M., Aziz, Z., Rauf, R., Shamsudin, N., & Latiff, R. (2019). TeBook a mobile Holy Quran memorization tool. In *Second International Conference on Computer Applications & Information Security* (pp. 1–6). https://doi.org/10.1109/cais.2019.8769472

AbuZeina, D., & Alsaheb, M. (2013). Capturing the common syntactical rules for the Holy Quran: A data mining approach. In *Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences* (pp. 670–680). https://doi.org/10.1109/cais.2019.8769472

AlKhatib, H. H., Mansor, E. I., Alsamel, Z., & AlBarazi, J. A. (2020). A study of using VR Game in teaching tajweed for teenagers. In *Interactivity and the Future of the Human-Computer Interface* (pp. 244-260). https://10.4018/978-1-7998-2637-8.ch013

Alotaibi, F., Abdullah, M., Abdullah, R., Rahmat, R., Hashem, I., & Sangaiah A. (2018). Optical character recognition for Quranic image similarity matching. *IEEE Access* 6, 554–562. https://doi.org/10.1109/access.2017.2771621

Bin Musa, M., Niyaz Bin Yusop, M., Sopee, M., & Ali, M. (2018). i-Tasmik mobilepPlatform – Enabling tahfiz student to memorize Al-Quran independently. In *International Conference on Information and Communication Technology for the Muslim World* (pp. 24–29). https://doi.org/10.1109/ict4m.2018.00014

Boussenan, R. (2013). Electronic Management of Quran Sites. In *Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences* (pp. 700–723). https://doi.org/10.1109/nooric.2013.107

Elliott, J., Eckstein, R., Loy, M., Wood, D., & Cole, B. (2002). Java swing (2nd ed.). O'Reilly Media.

Elmarhomy, G. (2020) Understanding hidden meanings of Quran's words using mobile/Android application. *Journal of Computer and Communications*, *8(9)*, 41-53. https://doi.org/10.4236/jcc.2020.89004

Hakak, S., Kamsin, A., Tayan, O., Idris, M., Gani, A., & Zerdoumi, S. (2017). Preserving content integrity of digital Holy Quran: Survey and open challenges. *IEEE Access* 5, 7305–7325. https://doi.org/10.1109/access.2017.2682109

Ibnu Al-Jazari, M. (2003), *Sharh Tayibat Annashr Fee Al-qiraat Al-ashr* (*Interpretation of Tyyibat Annashr in the ten Readings)*. Scientific Books Publisher.

Ishak, S., Zaki, Z., Mohamad, K., Bahrin, M., Roni, N., & Musa, M. (2016). MyQiraat: An interactive Qiraat mobile application. In *Fourth International Conference on User Science and Engineering* (pp. 35–39). https://doi.org/10.1109/iuser.2016.7857930

Khadangi, E., Fazeli, M., & Shahmohammadi, A. (2018). The study on Quranic surahs' topic sameness using NLP techniques. In *Proceedings of 8th International Conference on Computer and Knowledge Engineering* (pp. 298–302). https://doi.org/10.1109/iccke.2018.8566248

Khan, N., Ahmad, N., Beg, A., Fakheraldin, M., Alla, A., & Nubli, M. (2010). Mental and spiritual relaxation by recitation of the Holy Quran. In *Second International Conference on Computer Research and Development* (pp. 863–867). https://doi.org/10.1109/iccrd.2010.62

Larbi, L. (2013). Voice search in the Holy Quran. In *Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences* (pp. 413–418). https://doi.org/10.1109/nooric.2013.86

Mahmoud, M., & Hassan, I. (2013). Artificial intelligence techniques for extracting individuals recitation of the Holy Quran from its combinations. In *Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences* (pp. 292–297). https://doi.org/10.1109/nooric.2013.65

Majdalawieh, M., Marir, F., & Tiemsani, I. (2017). Developing adaptive Islamic law business processes models for Islamic finance and banking by text mining the Holy Qur'an and hadith. In *IEEE 15th International Conference on Dependable, Autonomic and Secure Computing* (pp. 1278–1283). h t t p s : / / d o i . org/10.1109/dasc-picom-datacom-cyberscitec.2017.205

Menacer, M., Arbaoui, A., & Mogbel, A. (2013). Concepts and architecture of a dedicated virtual learning environment for Quran and its sciences. In *Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences* (pp. 362–365). https://doi.org/10.1109/10.1109/nooric.2013.76

Mohamed, E. H., & Shokry, E. M. (2020). QSST: A Quranic semantic search tool based on word embedding. *Journal of King Saud University - Computer and Information Sciences.* https://doi.org/10.1016/j.jksuci.2020.01.004

Priatna, T., Nurhamzah, N., Suryana, Y., & Nurdiansah, N. (2020). Developing management of Quran memorization institutions through the web system. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(1), 465–468. https://doi.org/10.30534/ijatcse/2020/63912020

Qayyum, A., Latif, S., & Qadir, J. (2108). Quran reciter identification: A deep learning approach. In *Seventh International Conference on Computer and Communication Engineering (ICCCE)* (pp. 492–497). https://doi.org/10.1109/iccce.2018.8539336

Rafi, M., Khan, B., Usmani, A. W., Qasim, Z., Ali, S., Hasan, S. O. (2019). Quran companion - A helping tool for huffaz. *Journal of Information & Communication Technology*, *13*(2), 21–27. http://jict.ilmauniversity.edu.pk/journal/jict/13.2/5.pdf

Rashid, N., Venkat, I., Damanhoori, F., Mustaffa, N., Husain, W., & Khader, A. (2013). Towards automating the evaluation of Holy Quran recitations: A pattern recognition perspective. In *Taibah University International Conference on Advances in Information Technology for the Holy Quran and Its Sciences* (pp. 424–428). https://doi.org/10.1109/nooric.2013.88

Salem, M. (2003). *Fareedatu-Addahr fee Ta'seel wa Jam'e Al-qiraat Al-ashr* (Fareedatu-Adhar in rooting and gathering the ten readings). Albayan Al-Arabi Publisher.

Tanzil Documents. (n. d.). *Download Quran text*. https://www:// tanzil.net/docs/download

Tulba, A., & Awadh, Y. (2016). *Jam'e Alqiraat Al-Ashr min Turuq Tayyibat Annashr* (Gathering of the ten readings from Tayyabat Annashr ways). https://archive.org/details/sa71mir_gmail_201608

Yauri, R. A., Abdul Kadir, R., Azman, A., & Murad, M. (2013). Ontology semantic approach to extraction of knowledge from Holy Quran. In *Fifth International Conference on Computer Science and Information Technology* (pp. 19–23). https://doi.org/10.1109/csit.2013.6588752