ARCHITECTURAL MODEL FOR TEXT TO SIGN LANGUAGE INTERPRETER(TSLI)

Liyana Pathirannehelage Manupriya Banu Liyanapathirana

(199380R)

The thesis submitted in partial fulfillment of the requirements for the degree Masterof Science in computer science specializing in Software Architecture

Department of Computer Science & Engineering

University of Moratuwa

Sri Lanka

June 2022

To My Parents.

Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future work (such as articles or books).

Name of Candidate: L.P.M.B.Liyanapathirana

Signature: ..... Date: June 26, 2022

The above candidate has carried out research for the Masters thesis under my supervision.

Name of the supervisor: Prof. Indika Perera

Signature: ..... Date: June 26, 2022

Abstract

The deaf and hearing-impaired make up a considerable percentage of the world community having some essential needs that researchers have recently begun to target. There is no such proper way to communicate with deaf society instead of learning sign language. Sign language is a natural language that can be used as a powerful weapon for communicating with deaf society. The target objective of the research is to propose a Generic Architectural model to bridge the communication gap between deaf and non-deaf people. In the desertion, this problem was tackled by presenting the "Architectural model for Text to Sign Language Interpreter" system to translate text into any given Sign Language. Language parser Handler service is introduced as an API Contract. This Language parser Handler service acts as an abstraction layer for the backend services which are doing the NLP processing. Any researcher interested in translating sign language from a text can use this proposed model. A new language can be introduced to the model with minimum configurations. Instead of developing the entire model they just need to focus on the actual Language processing part. It should be a RESTFul microservice. This model can be used to test their API

In a nutshell, the Language-specific parser service does the following. First, the individual words are extracted by tokenizing the input sentence. The morphological analysis is then performed to identify the basic elements of the respective words. Basic components can be a sign stem, sign marker, or fingerspelling characters. After that, the identified basic word components (stream of tokens) are sent back to the client-side as a JSON string.

The primary tokens are extracted from the JSON output. Then the tokens are mapped with the corresponding SiGML files which are forwarded to the signing avatar embedded in the client UI. The final output is made by the signing avatar model which gives a stream of sign frames.

keywords: Sign Language, Service-Oriented Architecture

Acknowledgments

I would like to thank and extend my heartfelt gratitude to Prof.Indika Perera for the continuous supervision of my research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me throughout the research, and in writing this dissertation

Besides my supervisor, I would like to thank Mrs.Renuka Padmalatha who has given me knowledge of the Sinhala sign language, for her encouragement, insightful comments, and time consecrate for the research.

Special thanks to all the people who had done their research based on related areas and had committed their lives for the success of inventions. Without their contribution, this thesis would not be a reality.

My sincere thanks also go to my parents, my wife, and my family who had given me help and courage to do this research and to be strong in life as a successor. Also, I thank my friends who are always with me in any situation.

In order to preserve anonymity they cannot be named. Without their help, carrying out this research would not be possible.

Manupriya Banu Liyanapathirana, June 26, 2022

Table of Contents

List of Figures

Nomenclature

ASL American Sign Language

ASR Architectural Significant Requirments

ATAM Architecture Trad-off Analysis method

BSL British Sign Language

HamNoSys Hamburg Notation System

SigML Signing Gesture Markup Language

SL Sign Language

SOV Subject Object Verb

SSL Sinhala Sign Language

SVO Subject Verb Object

XML Extensible Markup Language

CHAPTER I

INTRODUCTION

The deaf in society has their own culture, which is very different from ordinary people. The main feature of this culture is that they use a different way of communicating with others, which is called sign language. Instead of using voice as a means of communication, they use gestures and facial expressions. Sign language is their primary language and other languages are secondary languages for them [1]. Unfortunately, the government and the people, in general, may not recognize it correctly. Human rights should apply equally to all people, including people with disabilities. But in reality, it is not considered to be important. People in developing countries like ours are the ones who suffer the most from this problem.[2].

Deaf people face many difficulties during communication with each other. According to the statistical results, they face trouble in learning natural languages. Furthermore, understanding a written language may also be hard in some situations[3, 1]. It is very difficult to learn secondary languages for a particular deaf person due to certain reasons such as, learning a second language means memorizing written symbols and mouth patterns which are highly ambiguous[1]. As a result of that, most of the deaf people at age 16 are facing difficulties in reading and understanding the grade 4 level textbook or higher one than a non-deaf person[4]. Moreover, the reading level of deaf students around age 18 is the same as the reading level of non-deaf students at the age of 10[1]. Many deaf students leave schools due to reading and writing issues[1]. Most of the time, the problem is related to communication. However, expert knowledge can be used to help them to increase their

lifestyle.

Three domains are mainly supported to develop the text-to-sign language interpreter such as natural language processing, knowledge of sign language, and 3D avatar libraries development[3]. Sign language has so many variations when comparing different countries, sometimes it varies from region to region also[5]. But after the 1950s sign language area was enlightened since arise of sign language studies.

An Architecture for Text to sign language interpreter is being proposed in this research. Without translating a natural language into sign language manually, it should be possible to translate that automatically by a computer so that deaf people can understand it. First of all, the meaning of a particular sentence should be identified to extract the structure of the sentence. Then, identified meaningful tokens are being mapped with corresponding signs. Furthermore, the feasibility of the assessment level of comprehension is addressed.

Furthermore, this research should be feasible for any deaf or non-deaf person who is interested in learning sign language. If someone has deaf friends and facing difficulties with communication, this research outcome will help them to overcome their problems. Not only that but also parents who have deaf children will also be benefited from this research.

Research Question: "How feasible is it to use the Microservices Architectural model for text into multiple sign language translation?"

1.1 Motivation

At present, numerous existing *"Text to sign language interpreters"* are available for different sign languages. Assume a research student has developed a text-to-sign translation library. To evaluate the new sign language library, they have to spend time and effort on developing a prototype that is totally out of the scope of their research topic. If there is a prototype available and its architecture supports any

sign language library with minimum configuration changes, then researchers can use it as a model and save their time and effort for the actual problem they are working on. This is the main motivating factor in this research.

The major problem faced by deaf people in dealing with society is the difficulty in communication with normal people due to the deficiency of a proper mode of communication and due to lack of a proper interpreter between the normal languages and the sign language. Thus most normal people are demotivated to make relationships with deaf people because they become uncomfortable in the process of communication with deaf people. This has become a major issue for the normal parents of deaf children because they need to learn sign language if they are to communicate with their children properly. But learning a new language is not easy at all, especially for adults. This is far more difficult when it comes to Sign language, as there is no proper guidance or facilities to learn it. Most of the time, the issues of deaf people are missed by society especially due to the above-mentioned communication barrier. Also, normal people do not pay special interest in learning sign language as they do not feel it to be important. They can not be forced to learn sign language as a solution for the mentioned problem. Those problems can be taken as motivation factors for this research.

News programs are broadcasted for deaf people in many countries. Normally, the news is read by a person and it is translated into sign language in real-time by another person who knows both sign language and natural language well. This whole process is done manually. However, there are several problems behind this scenario such as an additional expert person who can translate the natural language into sign language is needed. Most of the time both the newsreader and the translator have to practice the program many times for synchronization. Moreover, the whole translation process is a time-consuming one. Therefore, problems inside a news telecast for deaf people, have motivated me for doing this research.

Teachers who teach sign languages for deaf children should have to learn sign

language before they teach them. Most teachers are non-deaf people. Therefore, they have to put a big effort to learn sign language on behalf of innocent children who have hearing problems. However, there is no proper way of learning sign language. Moreover, teachers have no better learning materials in Sri Lanka. However, *Text to sign language interpreters* can be used as a learning platform for sign language learners.

The next major issue faced by deaf people is the problem of exchanging ideas with normal people in day-to-day situations, especially with service providers, for example at a post office or when buying something at a supermarket. Although normal people may understand the hand gestures of deaf people, they face difficulty in responding to the gestures as they have no clue of sign language. Thus a sign language interpreter is very useful to overcome this problem.

There is an eye-opening incident I experienced on a bus that made me think about this research topic. A deaf person on the bus who wanted to buy a ticket to his destination faced real difficulty in communicating with the conductor for this. The conductor asked the destination of the deaf person repeatedly both verbally and using the gestures he knew. The deaf person also tried hard to express himself using the hand gestures of sign language that the conductor could not understand. The other passengers of the bus also failed to help in this communication. So this problem would not have arisen if there had been a sign language interpreter in place.

1.2 Aim and Objectives

Aims:

- The main aim of this research is to build a generic Architectural model to support Text to Sign Language Translation.

- The suggested Architecture should be used as a framework to translate text

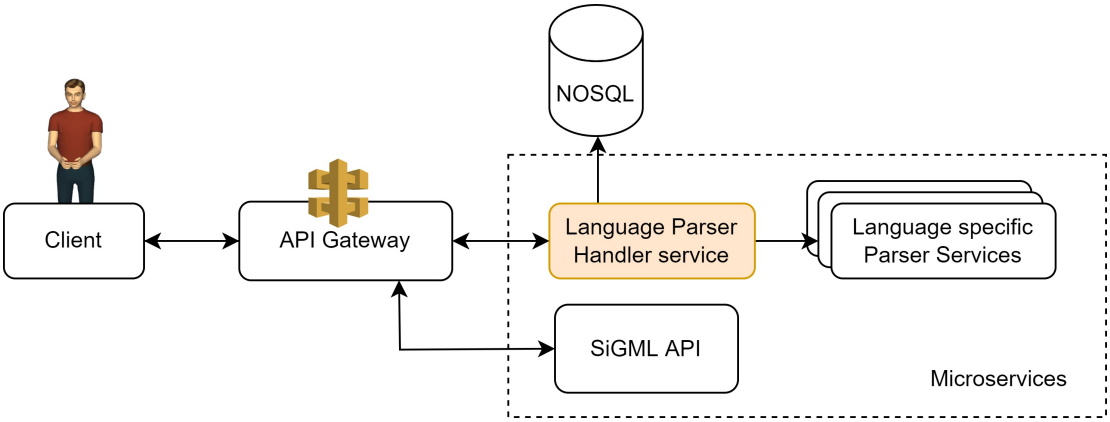


Figure 1.1: High level Architecture of the proposed system

to any sign language with minimum configuration changes.

Objectives

- The primary objective of this research is to assist researchers/students working on text into the Sign language translation domain. The framework proposed here can be used to test and demonstrate sign language translation libraries. They will be able to test/demonstrate their research findings using this architecture model without having to spend time and effort developing a prototype.

- Indirectly, the outcome of this research can be used to full fill the communication gap between deaf and non-deaf people.

1.3 Scope

The proposed model is developed based on Microservice architecture. The *Language parser handler service* acts as an abstraction layer to the Language specific parser services (Figure 1.1). Through this API contract, any service can be registered in the system without affecting other components.

In nutshell, the Input sentence will be translated into sign primitives also called tokens. If a particular token is not available in the library, then those missing words are shown using "finger-spelling" gestures[6]. Finger-spelling is a kind of alphabet which is used to spell words that are not in sign vocabulary. Each character in the sign alphabet has a different sign called fingerspell. Finger spells are utilized in this research to spell words that have no pre-established sign.

The anticipated result of this research is an interpreter which is developed based on the proposed architecture, that can translate Text into multiple sign languages. This is an application that uses text sentences as an input to generate appropriate sign language. Then the corresponding 3D gesture will be shown on the interpreter as an output.

- **Limitations**: Only one sentence can be provided as input at a given time. Only simple grammar patterns are acceptable as the input text. Therefore, very long sentences are not going to be considered in this initial model. The output of the text to sign language translation can be shown using a third-party 3D avatar library. Smoothening the transition between consecutive signs is a huge problem. The difference between the transitions of two sign movements is called coarticulation. Coarticulation may occur when interpreting the sign language using a 3D avatar library[7].

- **Out of Scope:** The implementation of the natural language processing module for each language is not part of this research. To accomplish the functional flow, existing NLP projects have been used to develop the proposed prototype. The base text parsing library is taken from the ISL project [59]. The Implementation of the SIGML generation module is not focused on this research as well. Pre-generated SiGML file libraries that have been taken from existing projects have been used for demonstration purposes. Indian sign SIGML file library is taken from the "Text To Sign Conversion"

project [60]. Example SIGML file set of BSL is taken from CWASA website [48]. For complete communication with the hearing impaired community, two-way communication is required. One is to convert text to sign language. Another major part is converting signs into text or voice. Converting signs into text involves computer vision technologies to capture the gestures and convert them into text. This can be done through several techniques, but a famous way of doing it is to use machine learning techniques. Converting signs into text is not considered in this research and will be considered in future work.

1.4 Structure of the thesis

The rest of the thesis is organized as follows. Chapter 2 is mainly focused on the literature review. In that section, three major domains are considered, natural language processing, knowledge of sign language, and 3D avatar libraries development. Architecture and the model of the text to sign language interpreter are discussed in Chapter 3. In chapter 4, experiments and evaluations are conducted. Two major evaluation procedures and corresponding results are analyzed in this chapter. Finally, chapter 5 concludes the thesis, and future work is mentioned.

CHAPTER II

LITERATURE REVIEW

2.1 Deaf culture

Most of the people tend to notice deafness as a fault. It is a culture which creates different dimension of communication. The voice is seen instead of hearing by deaf people. The deaf can be defined as a group of people who cannot hear human speech under any conditions[6]. According to the Helen Keller's view, she said that "Deafness cuts people off from people"[6]. The real side of the deafness is revealed by her. Reality is deaf people are being isolated in the society. Communication problems are being faced by deaf people. Unfortunately, people are thinking signing is an inferior substitute for actual communication. Furthermore, people assume all deaf people have the ability of reading lips. However, reading lips and understanding the meaning is not happened in deaf culture. The sign language is the powerful solution for the deaf communication.

The life is tried to start by deaf child without the normal associations with the hearing world to a degree which is not yet investigated[6]. Sound and hearing are very important to boost up first moment in the life. A normal hearing child utilizes significant time period within the first four weeks of life responding to sounds[6]. After end of first year, the verbal is tried to understand by normal hearing child. At year three, four and five stages of a normal child, development of logical expressions in words and sentence structuring, asks 'why' questions, try to discuss more remote and difficult problems respectively[6]. Unfortunately, those experiences are cut off from the deaf child. Furthermore, parents do not know

about the deafness of their child until the child is two or three years of age.

2.2 Sign Language

Signing is a language that utilizes gestures and facial expressions to communicate a meaning instead of using voice patterns to convey a meaning. In another way, sign language is a visual-gestural language which is utilized within deaf communities can express any function fulfilled by natural languages[8]. Signing is very useful for people who cannot speak though they deaf or not. However, the deaf people are addressed in this research. Sign language is enabled people to communicate via visual input channel with a gestural output channel. Therefore, deaf people have to deal with concurrent information while using different kinds of articulators[1][8]. Space which is utilized by signer[2] is called 'signing space'. Signing space is used intensively in any sign language is the specialty of it[15].

There are two major techniques for animate a virtual signer[8].

- Pre-synthesized animations – Animations can be pre recorded video clips. It is selected according to the input context. At present, this concept has been introduced to build sign libraries. That library is very useful for generate sign language utterances [16].

- Generated animations – Normally, this technique is used to generate animations automatically in real time. It can be achieved by using symbolic description languages such as Sign Markup Language called SigML(subsection 2.3.3). SigML is supported to define sign sequence using XML. Animations are constructed by many applications according to the descriptions of signing movements in SigML. Signing motions belong to "Gestural-SigML" which

[1] Phonetics any vocal organ that takes part in the production of a speech sound. Such organs are of two types: those that can move, such as the tongue, lips(http://www.thefreedictionary.com)

[2] a person expressing, using sign language

is a subset of SigML notation. Furthermore, those notations are based on HamNoSys (subsection 2.3.3) notations[3].

Same as the spoken languages sign language is a combination of set of phonemes[9]. It has a complex grammar and can be discussed any subject, from simple to complex. There are some relationships with natural languages. Most of the time, gestures are used to spell the words. If there are no sign for a word, then they use the concept of "finger spelling" to convey the meaning[5]. Finger spelling is the alphabet of sign language as normal languages. Each and every character has a gesture to indicate the meaning.

2.2.1 Sinhala Sign Language

SSL is a linguistically under investigated language with lack of well documented e-data while comparing with ASL or BSL. Sign language using in Sri Lanka is called Sri Lankan Sign Language. However, sign language in Sri Lanka is differed according to the race; both Sinhala and Tamil people have their own sign language which is derived according to the mother language of their race. According to the research paper [10] there is an argument that India, Bangladesh, Pakistan, and Nepal are using the same sign language. SSL has its own syntax, grammar, phonology and morphology. In ASL, the corresponding signs sequence for the text "What's your name" would be *YOUR NAME WHAT*[1]. According to the example, the sign for the 'name' in ASL is transcribe as NAME. N-A-M-E is a combination of series of linguistic symbols which have no relationship with the actual form of the sign[11]. However, in SSL, sign sequence would not be change with the corresponding question. Figure 2.1 shows similar sign sequence for Sinhala sentence. The sign sequence is corresponded to the Sinhala text *OBAGE YAHALUWAN KAWRUNDA* (obəge: jahaluvan kavrunðə ?)[24]. Sign order and the word order in the text is same according to the figure 2.1. Specialty of Sinhala sign language is, order of words in a given input text will be remained in same

order when convert into SSL. Rule base translation is a widely used technique in many related types of research and, it is used to convert natural language into Sign language [25, 27].

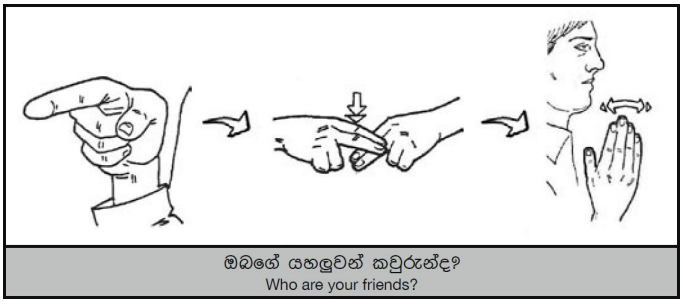


Figure 2.1: Corresponding sign sequence for *Who are your friend*s [24]

2.2.2 Sign language features

Sign languages have many similarities when comparing with verbal natural languages. Signer can utilize three dimension nature of the space in sign language which is not available in verbal natural languages. In many cases, Sign languages may have similarities to its geographical local verbal languages though it has no direct relationship with its geographical verbal languages [12]. However, there is no global sign language or standard way of signing. It differs from country to country and even it could have some variations from region to region within the same country[17, 5, 18]. Sign languages have their own syntax and grammar. Therefore, it can also be classified as natural languages [10]. In any sign language have common features which are described below according to the BSL[12].

2.2.2.1 Sign Order

Topic-comment structure has been used in BSL. Hence, the topic or main informational idea is signed first. The rest of the sentence or phrase is a comment, after identified the topic. Specialty is BSL has no ridge order of basic elements

such as SVO[12]. However, Sinhala sign language is normally used a grammatical structure like SOV. Same as Sinhala language, order can be changed according to the signers' view in Sinhala sign language.

2.2.2.2 Signing Space, Placement and Pronouns

In many sign languages, signers utilize signing space which is in front of the body. Speech components in sign language can be situated in that space. First, signer has to define the area and then all components are connected to that area. BSL contain more pronouns than English. Therefore, BSL has the power of expressing several variations of a pronoun using different sign. As an example it has the power of expressing plural pronouns such as *WE-TWO*, *WE-THREE* [12]. In SSL, pronouns are very similar with Sinhala language. Hence, two separate signs are used in SSL to represent word 'we-two'.

2.2.2.3 Directional or Agreement Verbs

Directional verbs comprise the facts regarding the subject, verb and the object. Normally, it can be done by moving the verb in virtual space when the subject and the object are positioned around the person is signing [12]. In SSL, most of the time, verbs begin at the object according to the SOV pattern.

2.2.2.4 Classifiers

Classifiers are kind of gestures which are used to classify an object into semantically related group. In BSL and ASL, classifiers can be utilized while signing. As an example, Tense of a particular verb in European SL can be exploded to the listener by using classifiers. Furthermore, classifiers are used in many sign languages to denote the group of an object.

2.2.2.5 Time Lines

BSL has no tense system[12]. In BSL temporal information are not expressed by using morphological or syntactical features connected with verbs. However, four time lines are used to express the time features.

2.2.3 Sign Language Phonology

Normally sign languages are utilized face, arms, hands, head and body postures to interpret a particular sign. 3D space around the signer is a vital factor in any sign language. A word in a spoken language is comprised with phonemes. A sign in sign languages is comprised with cheremes and two signs are differ by at least one chereme[3][10]. Chereme can be divided in to two parts, manual and non-manual cheremes[10]. Manual chereme has several parameters which change the behavior of a sign. Such as,

- Hand shape
- Hand location
- Orientation
- Movements

Same as that, non-manual cheremes can be defined as,

- Facial expressions
- Body or head posture and Eye gaze[10]

[3] chereme is a Greek word means 'hands'. It is equivalent to the phonemes of spoken languages

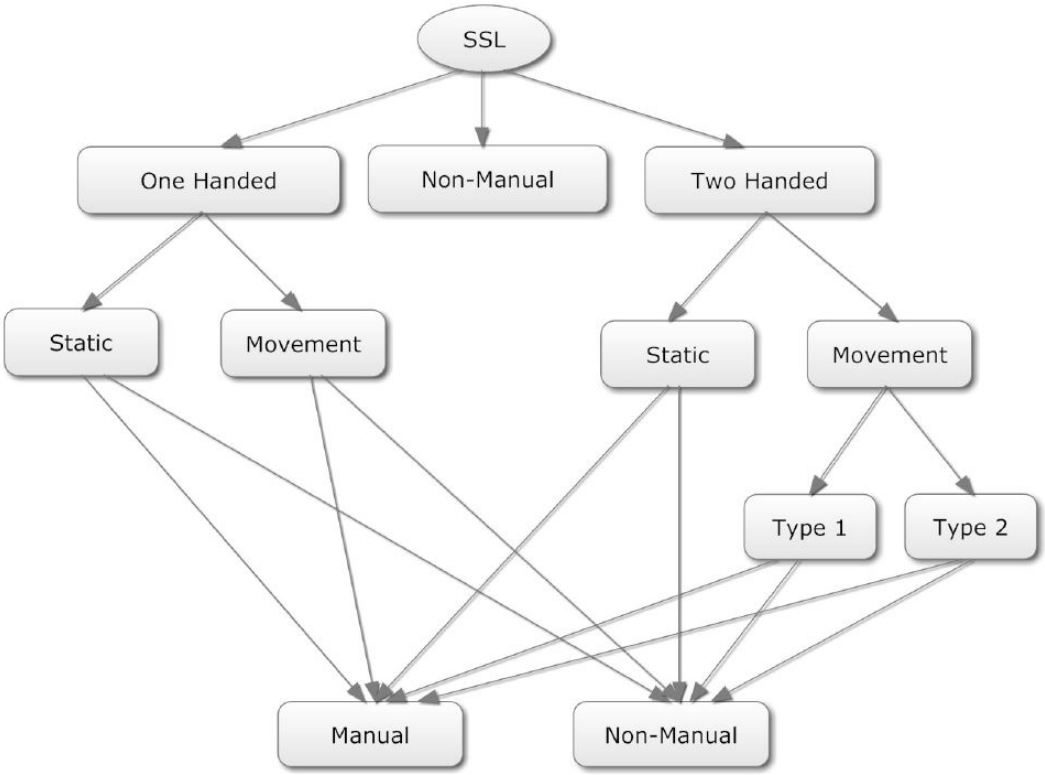


Figure 2.2: Basic categorization of Sign Language

Examples in Sinhala sign language are discussed in the following section to explain Sign language Phonology. Some signs in SSL which convey different meaning are very similar. However, the only factor that can distinguish between two signs is the sign motion. Some signs have the same hand shape, hand location, orientation and movement except non-manual cheremes like facial expressions. For an example, signs for *DUWANAWA*[4] (ðuvənəva:) and *HAYYEN-DUWANAWA*[5] (hajjen-ðuvənəva:) in SSL are got the same parameters of manual cheremes. Many people are struggling with the same questions, and how to identify the meaning of pair of signs? which portion of the mark adjust the meaning?[14]

The fifth basic component is the non-manual signs[14]. Considerable amount of

[4] *DUWANAWA* is present tense form of *run* in Sinhala

[5] *HAYYEN-DUWANAWA* means *running speedily*

signs in SSL need a non-manual part so as to convey the meaning correctly. Non-manual signals are the facial expressions that comprise with different signs. For example, the sign NEHA (næhæ) created with the open mouth and the shaking head with the tongue slightly out. Without these non-manual signals, the correct meaning is not conveyed.

Moreover, SSL signs can be divided into three categories; signs that uses a single hand, signs that uses both hands and Non-manual signs [10]. Categorization of the SSL is shown in figure 2.2. According to the attribute of the SSL, some signs contain either manual components or non-manual components and some are contain both of manual and non-manual components.

- One Handed signs: Signing is done only with one hand to interpret a certain sign. It has two properties static and movement. Static means posture of hand is not moving to utter a sign. Movement means posture of hand is changing while signing a particular sign. Each static and movement sign can be categorized into manual or non-manual elements [10].

- Two hand signs: Both of hands are used to interpret a particular sign. However, this category can be further categorized into two types; type 1 and type 2 [10].

  - Type 1: Both hands are active to interpret a sign
  - Type 2: Both hands are used to interpret a sign. However, only one hand is more active while comparing with other hand.

2.2.4 Sign Alphabet

Any sign language utterance consists of both signs and finger-spelled words. Same as that, SSL also contain the both. Two signs convey different meanings; structurally differentness helps to separately identify the meaning of a given sign. Signs

are kinds of morphemes. Ideas are generated with the concatenation of morphemes. Same as in Sinhala language, one or more morphemes are needed to express a unique word.

Some semantic concepts are aligned into a specific sign.

Figure 2.3: Sinhala sign alphabet

However, it is not possible to create signs for every word in the world. Words are rapidly added to any language day by day especially proper nouns. To overcome that problem, every sign language has its own mechanism called finger-spelling. It is an alphabet which has signs to utter phonemes. In other words, characters in a given word are spelled using signs in sign alphabet. Almost every character in Sinhala language has a distinct sign. A finger-spelled Sinhala word is a chain of digital symbols that alternate a one to one correlation with the letters of the Sinhala language. However, a word is a morpheme or a series of morphemes derived from Sinhala language sounds. It is important to know that, signs for alphabet are

developed according to the different sounds instead of mapping to all the characters in Sinhala language. Thus some characters are dropped in sign alphabet. Sinhala sign alphabet is shown in figure 2.3. SSL characters in sign alphabet can be signed using one hand [13]. ASL alphabet is one handed and the BSL alphabet is used both hands to sign a character [14, 19]. ASL manual alphabet is shown in figure 2.8.

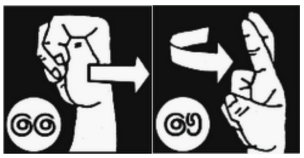


Figure 2.4: Minor vowel signs for sound *ai* and *au*

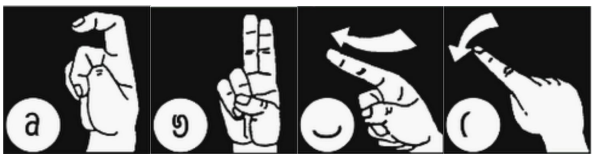


Figure 2.5: Phoneme modifiers in SSL

Though Sinhala language has 54 letters, SSL has 56 distinct signs. Reason for that is, there are two phonemes which are not actually letters. However, they are combined with other letters according to phonology. Thus, it is never going to be finger-spelled [22]. SSL contain 12 vowels, few minor vowels (see figure 2.4), small number of phoneme modifiers (see figure 2.5) and 38 constant signs. In Sinhala, most of letters can be paired together according to the sounds which are approximately the same sound. That is the main reason for similarity of many signs. However, some signs have twisting motion additionally to utter the letter difference as shown as in figure 2.6 [22].

SSL has twelve major vowels though Sinhala language has fourteen vowels. Furthermore, there are few minor vowels, small number of phoneme modifiers

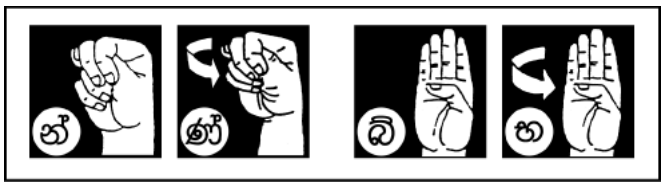


Figure 2.6: Different letter pairs in SSL

and forty constant signs [22]. The sounds "au" and "ai" which are dropped in alphabet are uttered combining two characters. Sinhala sign alphabet has the power of uttering any Sinhala word. It is very healthy phone table have rich sound variations. It is impossible to generate signs for all kinds of different letters which are exceeded 1000 in Sinhala language. As a result of that, certain word is finger-spelled using it basic consonants and vowels. Word "sri lanka" can be spelled as in figure 2.7. It is used several combinations of consonant and vowels. Though the Sinhala word "Sir Lanka" has only four letters, there are eight signs are used to represent it.

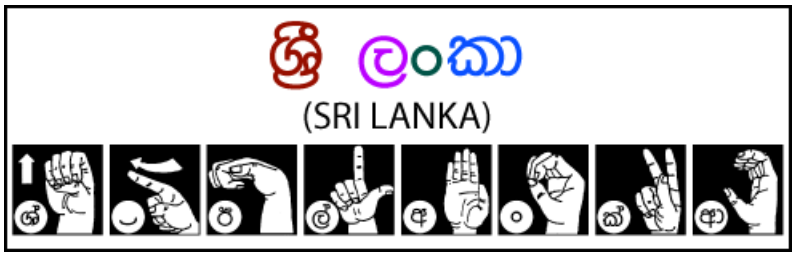


Figure 2.7: Finger-spelled signs for word *"SRI LANKA"*

There are similarities between *ASL* and *SSL*. There is an open discussion; *ASL* is utilized by people who invented Sinhala sign alphabet [22]. As an instant, the left pair of signs in figure 2.6 for letter 'N' is exactly the same sing in ASL for the same sound. Moreover, ASL manual sign for letter 'B' which is shown in figure 2.8 and same sound in SSL which is represented at right hand side in figure 2.6 is same [22]. However, most of sign manual alphabets have similar features.

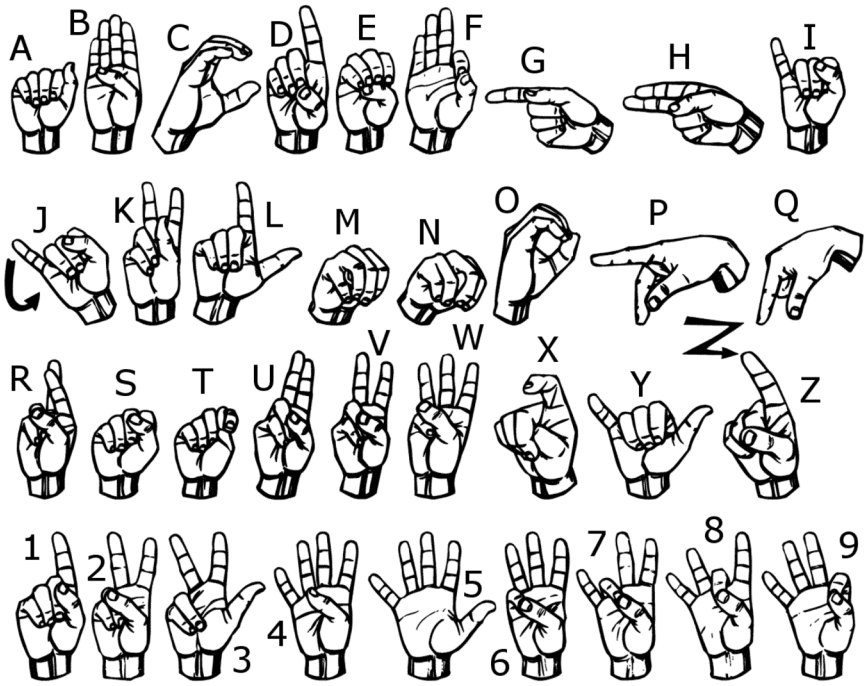


Figure 2.8: Finger-spelling alphabet for ASL

2.3 Existing systems

2.3.1 Speech to Sign Language Interpreter System (SSLIS)

According to the current status, researches have been conducted by many people in the world. Authors In [5] describe a methodology to fill the gap between deaf and non deaf people. It mainly focused on American Sign Language. Software (SSLIS) which can translate voice in to ASL (American Sign Language) is developed by them. Furthermore, they used several tools to develop such a system using Sphinx 3.5[6] for voice recognition, 3D avatar for represents the signs. In this project, finger-spelling is used to show unknown words which are not in the library. Input text

[6] speech recognizing tool which have trainers, recognizers, acoustic models [5]

will be recognized by the system and it will input to ASL database which perform word matching. ASL database have prerecorded basic sign video clips. Each basic word has a recorded sign clip [5]. If there is a match, system will be displayed a corresponding sign video clip. Otherwise, finger-spelled word is displayed by the system [5]. Figure 2.9 shows the basic structure of the Speech to sign language interpreter system [5].

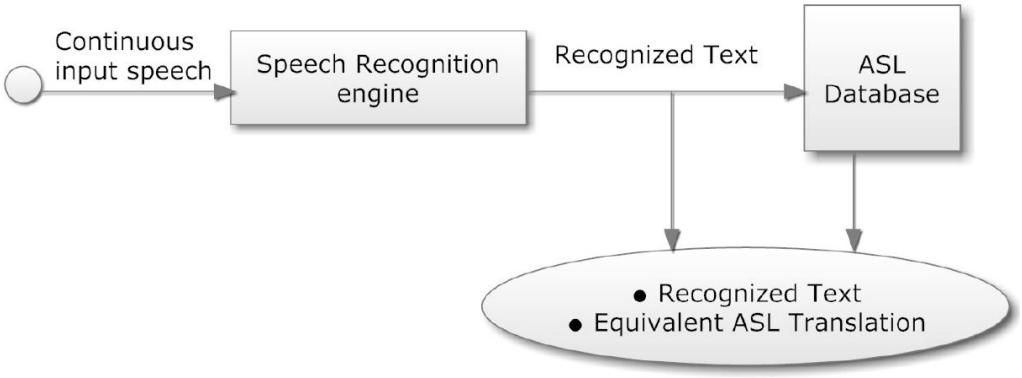


Figure 2.9: Structural design of speech to sign language interpreter

The specialty of the system is that, sign markers are used together with the sign words. Sign markers are already signs which are used to explicitly utter the category of a particular sign. As an example, tense of a sentence, person of a verb can be shown by using sign markers. Either a sign word alone or a sign word and one sign marker together can be used to utter a English word [5].

2.3.2 Speech to Sign Language translation system for Spanish

Authors in [3] researched about Spanish sign language and developed a system. *ESign* 3D avatar is used to represent the signs. In that project rule-based translation and statistical translation is used for natural language processing. However, it has a considerable error rate when try to express a general nouns or unspecified nouns. First, instead of mapping the text directly to signs, an analysis is made between the semantic concepts and the signs to find a better relationship[3]. There

are four distinct situations considered when translating the Spanish language into Spanish sign language as follows [3].

- Each specific sign has a corresponding semantic concept.

  Semantic concept is directly mapped into a particular sign [3]. In other words, semantic concepts which are extracted from the text are assigned to a sign.

- Some semantic concepts are aligned into a specific sign.

  If there is a unique sign generated by several concepts then the semantic concepts should be unified [3].

- One semantic concept generates several signs

  If it is needed to produce more than one sign from a single concept, then the sign order and the sign sequence might depend on the concept [3].

- More than one semantic concepts produce more than one signs

  If it is needed to produce more than one signs from more than one concepts with relationships among them [3]. As an example, the word 'fly' can be represented using different signs according to the subject which is interested such as bird and plane [3].

Overview of Spanish to sign language translation is shown in figure 2.10

2.3.3 Existing 3D avatar tools

ViSiCast, eSign, JaSigning, CWASA are few virtual signing research projects. Most of the existing signing avatar tool accepts SiGML as the input sign language notation.

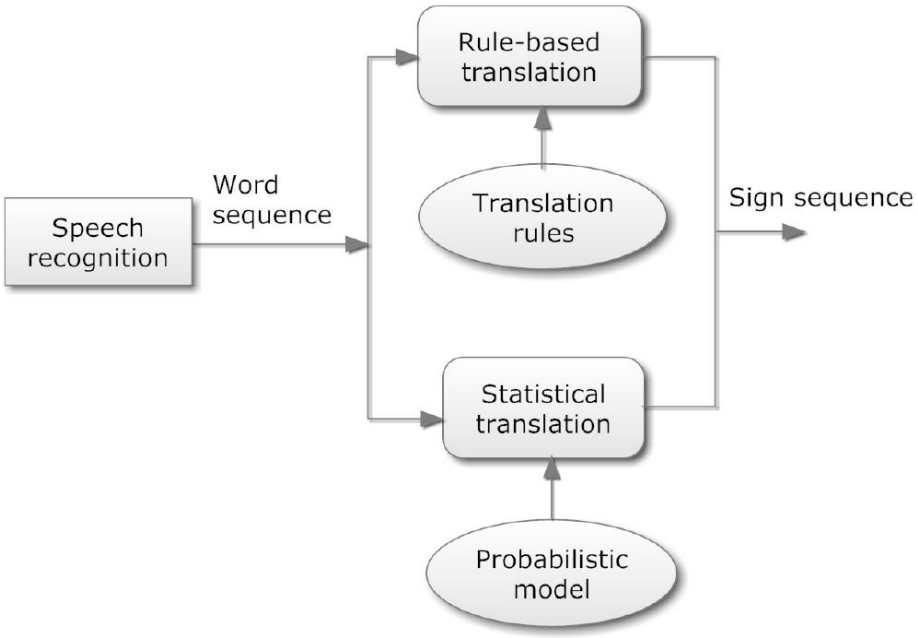


Figure 2.10: Structural design of Spanish to sign language translation

- **SiGML**: SiGML is an XML variant that is used to document the signing language gestures. SiGML is developed on top of HamNoSys technology. The University of Hamburg has developed a few SigML editing tools that can be used to generate SiGML files[50]. Number of SigMl edition tools can be found at [52].

- **HamNoSys**: HamNoSys also called the Hamburg Sign Language Notation System, was developed by IDGS of Hamburg University. HamNoSys describes signs on a 'phonetic' level. HamNoSys signs can be described using nonmanual features, handshape, hand orientation, and location [51].

2.3.3.1 eSign

"*eSign*" avatar plug-in is 3D model which capable of animating signs. This 3D model consists of a temporal series of frames. Every frame describes a posture at a certain moment. It should send sequence of pre-specified animations, in-order to

make an avatar sign [3]. Signed animation is automatically generated according to the *SiGML* format also known as Sign Markup Language[3]. *eSign* signing avatar is shown in left hand side of the figure 2.11.

2.3.3.2 ViSiCAST

ViSiCAST is funded as part of the EUs Framework V program which was a three year project [29][12]. It is also a 3D avatar library like *eSign* which can capable of making signs [7]. The main purpose of developing this project is to provide services and information access to deaf community. Several objectives were carried out through *ViSiCAST* project such as, various technologies were used to study sign language interpretation[20], possible speech to sign language conversion is studied in some selected domains [21], experimented on translating English text into multiple sign languages [12]. *ViSiCAST* signing avatar model is shown in the right hand side of the figure 2.11

Text to sign language conversion has two parts. First, input text is converted into an inter-lingua representation. Then after, inter-lingua representation will be translated into graphically oriented representation. It can be driven by such virtual avatar like *ViSiCAST* [12].

Figure 2.11: *eSign* signing avatar and *ViSiCAST* signing avatar

2.3.3.3 JASigning

JASigning (JAVA Avatar Signing) virtual signing system is mainly developed on top of Java NLP apps. Jasigning project replaced the SiGMLSigning model which was initially developed for ESign and ViSiCAST. Due to the lack of web browser support for Java apps, this project was deprecated[32].

2.3.3.4 CWA Signing Avatars

CWASA[46] is a virtual signing system developed by enhancing the JASigning system. HTML5 technology using WebGl is used to develop the CWASA project which is not dependent on Java like in previous JASigning solution [32]. One of the CWASA solution is implemented to fully operates on client-side [47]. CWASA can be installed in locally [48]. SigML player project is available at [33]. There are five characters available in the player. They are 'Anna', 'Marc' ,'Luna', 'Siggi', and 'Francoise'. This can be configured to run either on client side or server side. The configuration setup of CWASA for HTML5 web pages can be found at [47].

Figure 2.12: CWASA: Marc and Anna

CWASA has used a component called Animgen which converts the SiGML into gesture data that can be understood by any avatar[49].

2.4 Technologies

2.4.1 Python

Python is a high-level, object-oriented programming language that supports packages and modules. Therefore, reusing the existing code and, program modularity will be easy to accomplish. The latest python version 3.10.2, was released on Jan/14/2022 [53]. *PIP* can be used to manage packages and modules in Python. Missing packages can be installed using the command line with the PIP command.

2.4.2 Flask framwork

Flask is a lightweight micro web development framework which is built in python. Flask can be used for rapid web development without developing it from scratch. Flask has two main python libraries called Werkzeug and Jinja2. Werkzeug provides the web server gateway interface (WSGI), debugging and routing. Jinja2 provides the template support for the Flask server. Werkzeug and Jinja2 are by default authorized in Flask [34][35][36].

2.4.2.1 Installing python packages with Flask

Most common way to install python packages is using pip command. Following command can be used to install flask on a python environment.

```
$ pip install flask
```

flask can be imported in python code like below; During the Flask server initialization, it creates an application instance. All the client requests will be passed to the application instance via a WSGI protocol. WSGI refers to the web server gateway interface. __name__ variable of Python is passed as a parameter to provide the name of the main package of the application, during the initialization[35].

```
from flask import Flask
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
```

2.4.2.2 Routes and View Functions

app.route is used to identify the request and route to the correct function. In this example, the function paser_handler() acts as the application's handler for the /si URL. If the request came as the http://www.tsli.com/si from the browser, then it would trigger the paser_handler() function on the server-side response will be a document which the browser would understand[35].

```
@app.route("/si", methods=['POST'])
def parser_handler() -> str:
    #do the logic here
```

2.4.2.3 Server Startup

The following piece of code is used to start the server. After the server startup, it waits for requests for the client and the server continuously checks for the request within a loop[35].

```
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5001, debug=True)
```

2.4.3 NoSQL Databases

RDBMS is a strong tool for traditional data manipulation. However, the amount of data generated by the current applications is becoming very large. Those large data cannot be stored or processed in an RDBMS. NoSQL databases are mainly introduced to accomplish the **Big Data** problems. Most **NoSQL** databases offer high availability and horizontal scalability than relational databases[37]. The main advantages of NoSQL databases; are can be easily expanded, have fast read/write data access, have Big Data support, maintenance cost is low[41].

Based on the way NoSQL databases store and retrieve data, many NoSQL solutions fall into either Document store, Column-oriented , or key-value store [37][41].

- Key-value stores

Basically, this contains key-value pairs with Conclusiona unique key. Get and, Put are the only operations supported for this model. Simple CRUD operations are only supported by the pure key-value stores. Due to the above factor, the key-value store is called schemeless[38]. This is a very simple model and it allows to query data quickly. The database performance level is high due to the above

reason. This database model is not enough to cater to complex operations like range queries[37].

- Document store

Values stored as a JSON document. Accessing the data is more flexible than to the key-value store. Since data is stored as a JSON document, part of the data can also be fetched if necessary[37].

- Column-oriented

Data is separated stored for the column basis. Data in each column is indexed using the column. Support concurrent process to query data. This data model is more suitable for data warehouses [41].

2.4.3.1 Existing NoSQL Solutions

There are many open-source NoSQL databases available in the market. For example, MongoDB, Redis, HBase, CouchDB, Cassandra, etc. There is a tradeoff between Consistency (C), Availability (A), and Partition-tolerance (P) according to the CAP theorem.

- **MongoDB**: As per the qualitative comparison conducted in [37], MongoDB uses the Document type model for the data storage. It provides robust compatibility and shows a stronger ability for functionalities like scan queries. It is non-relational but provided more features like a relational DB. It supports complex data types like bjson. It has powerful querying features. Fast data access. It's 10 times the speed compared to MySQL when the data size exceeds 50GB[41].

- **Cassandra**: Cassandra is an open-source NoSQL distributed database which is licensed by Apache. It's hybrid: can be worked with no data loss even with

an entire datacenter outage, falt-tolerant: replication can be done across multiple data centers, scalable: when new machines are added read/write throughput increases linearly[39].

- **Apache HBase:** HBase is an open-source distributed NoSQL database which is also the Hadoop database licensed under Apache. This can be used to store big data. It's a highly scalable, reliable solution and supports high-performance computation on inexpensive commodity hardware. This can be used to manipulate big data with real-time access[40].

- **Redis**: Redis uses the Key-value pair model to store data. During the Redis execution, all the data is loaded in the memory. From time to time, data is saved to the hard disk in asynchronous mode. The architecture supports more than 100000 read/write operations per second. The main disadvantage is, that this model highly relies on the size of the physical memory. The size of the database cannot go beyond the size of the memory. Due to that factor, this is not a good solution for big data manipulation. This model is highly recommended for high-performance data processing of a small set of data[41].

- **CouchDB:** CouchDB is a P2P-based distributed database that provides fault-tolerant. Also, it is a flexible DB that supports REST API. JSON and AtomPub formats can be used to store data. Also, it provides bi-directional replication support[41].

2.5 Software Architecture Evaluation Techniques

Software architecture evaluation is very important to test if the proposed architecture has met the required quality attributes which are defined at the initial stage [55]. Specific quality attributes such as performance, reusability, scalability,

integrity, modifiability, security can be archived through an Architecture. However, since there are trade-offs between quality attributes, some quality attributes cannot be achieved together. Architecture describes its characteristics and risk in the design apart from the quality attributes [55]. Following are widely used architecture evaluating techniques. In this section, the main focus is on scenario-based techniques.

- Scenario-based: This evaluation is carried out to measure the quality attributes by creating scenarios [55].
- Experienced-based: This evaluation is carried out based on the previous experience of the developers [55].
- Simulation-based
- Mathematical-based

2.5.1 ATAM - Architecture Trad-off Analysis method

ATAM is a well-known *scenario-based* method that can be used to evaluate system architecture. It depends heavily on the quality attributes. Using ATAM, tradeoffs between quality attributes can be maintained. Also, potential risks in the system can be identified. The main advantage of this method is that it can be used by outsiders who do not know about architecture. ATAM evaluation can be done using three groups: designers, peers, and outsiders. They must be impartial in their evaluation.

Designers are the fort group of evaluators. Architectural designing is done by them.This group follows a repetitive hypothetical method of design. It includes requirements analysis, design, and design review.

Peers are the second group of evaluators. For example Clients, Product Owners, Project managers, Technical leads, Software Architects. The important fact

is, they are related with the project but not part of the taking design desitions.

Outsiders are the third group of evaluators. End uses, Support staff can involved in this evaluation.

This method helps to identify the following outputs.

- **Risks**: Identifying issues that can happen on some quality attributes due to problems in Architecture design decisions[56].

- **Sensitivity point:** Identifying the critical properties of the components in order to achieve a specific qualitative attribute[56].

- **Tradeoffs:** Identifying the design decisions that might be affecting multiple quality attributes[56].

ATAM method has nine step processes which is described below.

1. **Present the ATAM:** The ATAM process will be presented by the evaluation team. The context for the evaluation, expectation, procedures, outputs are included in the process[56].

2. **Present the business drivers:** In this phase, the business problem, the goals for the system, features and requirements of the system, project constraints, and scope are presented by the project decision-makers[56].

3. **Present the architecture:** The proposed software architecture has been presented by the system architect[56].

4. **Identify the architectural approaches:** The architecture patterns that are used system is evaluated in this stage. The main goal of this stage is as follows. The architecture documentation is analyzed. Questions can be raised if there are any things to clarify[56].

5. **Create a quality attribute tree:** In this step, a quality attribute utility tree is created and it depicts the detailed requirements of identified quality attributes. Firstly, the quality priorities are identified with the help of project decision-makers by analyzing the utility tree. Then the Architectural Significant Requirments (ASR) are identified using Utility tee. Next, the identified utility system is divided into Quality attributes. Iteratively doing the same process will provide more refined quality attributes which are particular to the system. Next, the ASRs are mapped with the identified quality attributes.ASRs can have a priority value like High, medium, low[57].

6. **Analyze the architectural approaches:** The goal here is to analyze the architecture with the help of the prioritized utility tree that has been created and abstract the way it addressed ASRs. The outcome of the above analysis will be helped to identify the risk and non-risk scenarios, sensitivity points, and trad-offs. Also, the abilities of the system and the impacts of design decisions can be identified with these steps[61].

7. **Brainstorm and prioritize scenarios:** In this step, Quality attribute scenarios are created by the evaluation team. Also, similar quality concerns of behaviors are identified and merged during the process. Next, the identified scenarios are prioritized by all stakeholders according to their importance. After that, the prioritized scenarios in the utility tree are compared with the list by the evaluation team. It is considered a proper match if the priorities in the utility tree are aligned with the priorities in the list. During the process, stakeholders may identify new scenarios. This is where the evaluation team identifies the risks of the system[56][61].

8. **Re-analyze the architectural approaches:** This is very similar to the previous analysis. Here it is used only the top five to ten scenarios in the

prioritized utility tree. then identify the ways to achieve each scenario by discussing with the system architect[56, 57].

9. **Present the results**: This is the final step. The evaluation results are analyzed and documented. Risk scenarios are classified. The outcome of the ATAM analysis provides risk and non-risks scenarios, tradeoffs, and sensitivity points. Additionally, the utility tree is delivered with the documents[61][56].

CHAPTER III

METHODOLOGY

In order to identify the potential scenarios to address the problems discuss in chapter 1, the design phase is a vital part of the research. The main aim of the research is to propose an Architecture to fill the gap between the hearing impaired and normal people using text to sign language interpreter. In this chapter, the design of the research and the methodologies used are discussed. The main approach to identify the system requirements and the functional behaviors in architecture was analyzed according to the information gathered through the literature survey in chapter 2.

3.1 Research methodology

Many research methodologies are being concerned during the research. In order to develop a prototype of text to sign language interpreter, text analyzing methodologies, text processing methodologies, text to sign mapping methodologies were carried out through the research. The majority of research was conducted to facilitate European sign languages like ASL and BSL. However, there are less research and well-documented data for the SSL. Therefore, many techniques related to the research are taken from ASL and BSL. Moreover, information was gathered about SSL from teachers who know both SSL and Sinhala languages. SSL grammar has many relationships with the Sinhala language, which can be utilized through the research. According to the literature reviewed in chapter 2, limitations and constraints were identified. The above factors are further discussed in this chapter.

3.2 Prerecorded videos and avatar library

Prerecorded videos or 3D signing avatar is used by many sign translators. However, there are pros and cons to both methodologies. The most frequent problem of the prerecorded video is that it cannot change after the creation. Thus it cut off the possibilities of using in dynamic situations. in addition, the cost is high and internal parameters like gender, clothes, lighting cannot be changed and video cannot be anonymously represented[1]. However, signing avatar has several benefits when compared with prerecorded videos; production cost might be low, parameters can be adjusted after the production, execution can be done anonymously [1].In another way, according to the state of art, signing avatar is still not accepted the prefect comprehensive level of understanding and it is laid under the 60% [1]. The main reason is facial expressions, body movements, and speed variations are dependent according to on the idea or the subject. It is very hard to carry all the parameter variations as a real signer by using a signing avatar. However, in this research CWA signing avatar[33] has been used.

3.3 System Design

This section is mainly focused on the design of the text to sign language interpreter. The design structure of research is shown in figure 3.1. Those design modules are described in detail in next sections.

Architecture is designed using microservices architectural style. Moduls are communicating with REST calls. In nutshell, there can be simultaneous requests from clients and those requests can be identified as translation requests. First, requests will sends to the language parser handler service and then it invoke the relevant service endpoints by reading a property file. To start the process, UI client passes the input text along with the selected language to the Client event handler. Client event handler generates a REST call and invoke the Language

parser handler. In between we can see the API gateway. Along with the REST call, Client event handler sends the parameter values as a Json string. Once the language parser handling service receives the request, It checks if the same input sentence is already requested recently. If exists, it will take the output Token-string from the NoSQL database. If the record is not existing, based on the selected language, the Language parser handler service routes the request to the corresponding microservice. This Language parser handler identifies the relevant microservice by reading the property file. Once the Client event handler receives the parsed string, it identifies the tokens of the preprocessed string and sends it to the SIGML converter. Based on the tokens, SIGML Convertor identifies the relevant SIGML files and forwards it to the UI client as an input stream.

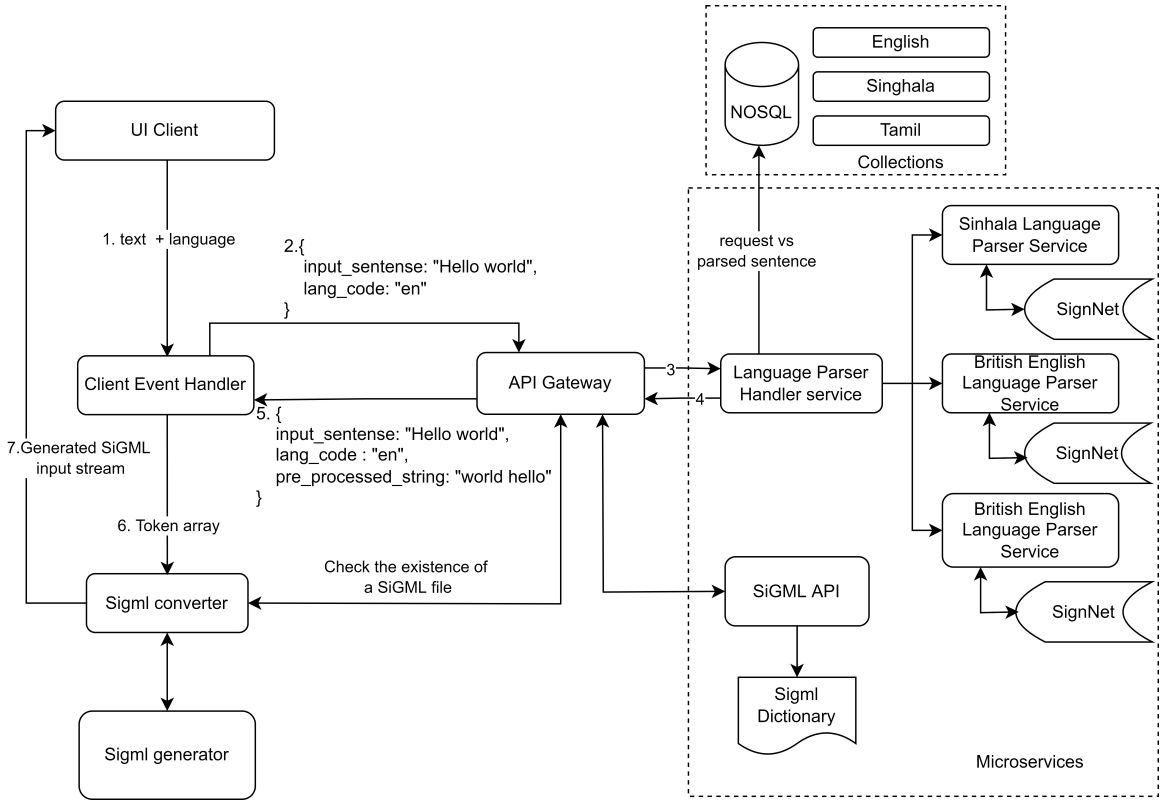


Figure 3.1: Architectural design of the proposed text to sign language interpreter

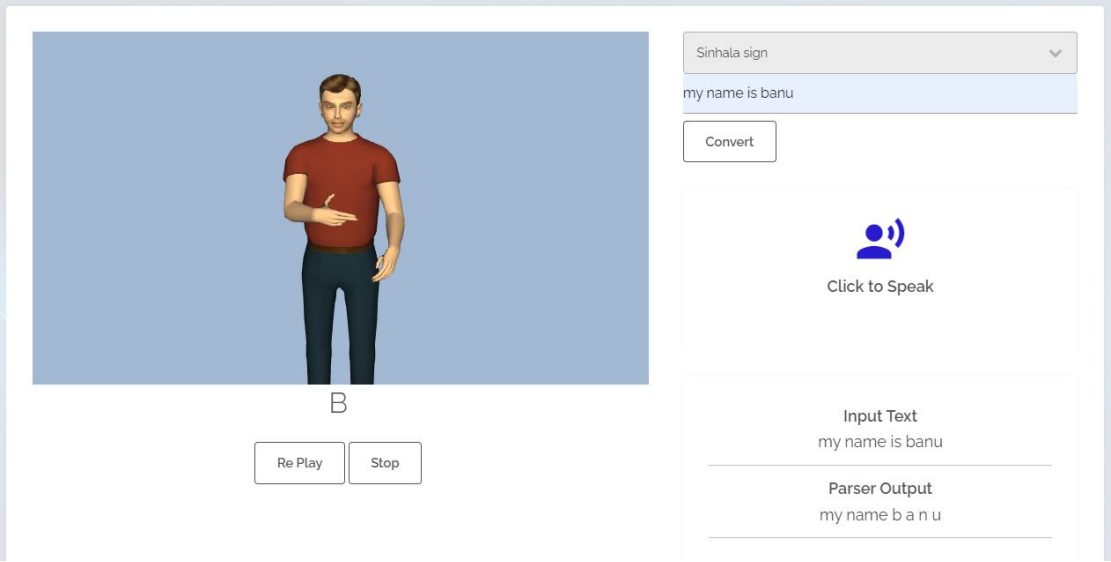


Figure 3.2: Client UI interface

3.3.1 UI Client

UI client is mainly based on HTML5 which consists of the following three technologies.

HTML which is used to build the structure of the page,

CSS which used for the presentation layer development and

JavaScript is used to develop the functionality of the UI Client.

The signing Avatar component is embedded into the Client UI. In this prototype, MARC (subsection 2.3.3.4) virtual human module is used to demonstrate the Architecture.

3.3.2 API Gateway

API gateway lays between the backend services and the UI client. this act as an abstraction layer between client and the backend services. It forwards the client requests to the relevant services which hide the underlying services from the client-side. Also, it enables the flexibility of code maintenance when code refactorizing

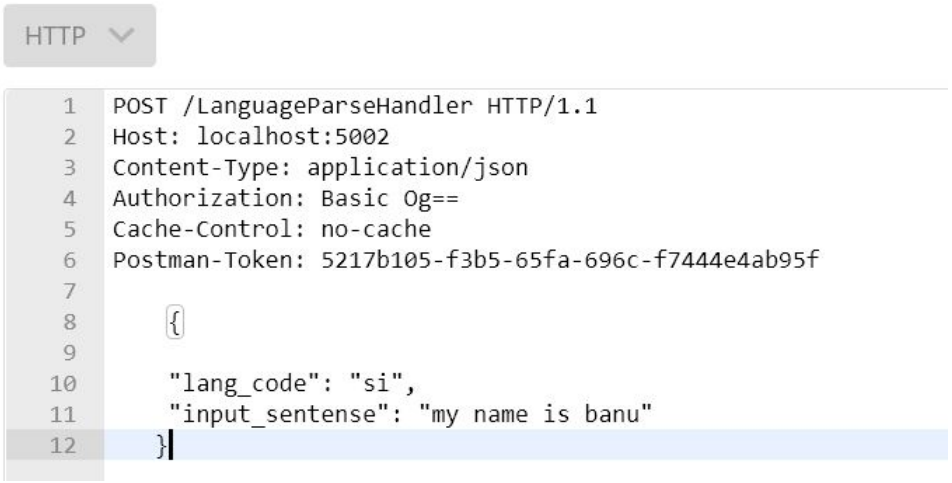


Figure 3.3: Invoking Microservices Language Parser Handler via Postman

is needed. However, it adds extra latency since the request should go through an extra routing point.

3.3.3 Language Parser Handling Service

This is s a RESTful microservices that is written in Python. This has been developed on top of the Flask server(subsection: 2.4.2). This service requires a JSON document as the input data along with the request. JSON payload of the request should be as follows (figure 3.5). The language code that needs to be parsed into and the input string that needs to be bound should be sent as a Jason string with the request. The same language parser handler RESTFul service can be invoked by the Postman tool[1]. Figure 3.3 shows what the Postman request looks like. The payload should be included in the body along with the request. Response of the *LanguageParserHandler* service is similar to the Postman output shown in Figure 3.4.

Parser Handler microService is called from the server inside the JavaScript code as sown in the figure 3.5. XMLHttpRequest library is used to generate the

[1] Postman is an application that allows the testing of APIs

Figure 3.4: Response of the LanguageParserHandler via Postman

request to the microservice. In the development environment, the Parser handler MicroService is listing to port 5002. It accepts all the POST requests that are come to the "/LanguageParseHandler" destination and reroute the request to another microservice base on the language code. Separate microservices are available for each language code to parse the sentence.

Before it reroutes the request to the language-specific microservice, it checks if the same input sentence has already been requested recently. To facilitate this feature, MongoDB NoSQL database instance ('signCache') has been created. A collection called "passed_string" is used to store input sentences against processed data for future reference. Data is stored as a document with information shown in the figure 3.6. If the processed data exists for a certain input string, the language parser Handling service responds to the request with the available pre-parsed data in MongoDB(subsection: 2.4.3.1). If a matching input sentence is not available, the handler service immediately calls the relevant language-specific microservice to process the input sentence. With the above-mentioned process, if a matching string is available for a particular input, then the latency of the sign generation process is less. If the input sentence is available in the collection, the language-parser microservice won't be called. Language parsing is a time-consuming service and the parsing time depends on the length of the input sentence.

Python library mongoengine[42] is used to connect to MongoDB from the

```
function getParsedText(speech) {
    let lang_code = document.getElementById('language_code').value

    var HttpClient = function() {
        this.get = function(aUrl, aCallback) {
            const params = {
                "lang_code": lang_code,
                "input_sentense": speech
            };
            var anHttpRequest = new XMLHttpRequest();
            anHttpRequest.onreadystatechange = function() {
                if (anHttpRequest.readyState == 4 && anHttpRequest.status == 200)
                    aCallback(anHttpRequest.responseText);
            }
            anHttpRequest.open( "POST", aUrl ,false);
            anHttpRequest.setRequestHeader("content_type" , "application/json");
            anHttpRequest.send(JSON.stringify(params));
        }
    };
    var client = new HttpClient();
    client.get('http://localhost:5002/LanguageParseHandler', function(response) {
        console.log(response);
        final_response = JSON.parse(response);
    });
}
```

Figure 3.5: Calling the Parser Handler microservice from JavaScript code

```
_id: objectId("62122da41d4c212a10a786ee")
lang_code: "si"
input_sentense: "My name is banu"
pre_process_string: " my name b a n u"
```

Figure 3.6: passed_string collection in mongoDb

```
from mongoengine import connect

#connect to the mongo db
connect(db='signCache', host="localhost", port=27017)
```

Figure 3.7: Connect with MongoDB from Flask

python microservice. Following pip command is used to install Mongoengine on python.

```
$pip install mongoengine
```

In the development environment, MongoDb runs at the default port 27017. To connect to the MongoDB within the Flask server, the code snippet in figure 3.7 is used.

3.3.4 Language parser services

For each language, there is a separate language parser microservice available to handle the request. There are multiple language parses that have been used in the development environment. For example, Sinhala language parser service, English language parser service, and Tamil language parser service. Each service parses the specific input sentence into a pre-processed sentence that can be used to identify the exact sequence of how the Sigml files should be executed on the avatar.

This service consists of the following four features. The prototype has been developed in python by using NLTK(Natural Language Toolkit [44]) python library.

1. **Parse the input sentence:** This module generates the parser tree of a given sentence. In this prototype, the Stanford parser [43] has been used. The parser tree of the Stanford parser for the input sentence “My name is Banu” is below. The syntactic structure of a given sentence can be identified by analyzing its underlying grammar. Stanford parser is one of the famous tools to generate the parser tree. Sentences consist of the noun phrase and

```python
from nltk.stem import WordNetLemmatizer

def lemmatize_tokens(token_list):
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = []
    for token in token_list:
        lemmatized_words.append(lemmatizer.lemmatize(token))

    return lemmatized_words
```

Figure 3.8: Lemmatizing tokens with python's NLTK

verb phrase.

```
(ROOT(S
        (NP (PRP$ My) (NN name))
        (VP (VBZ is) (NP (NN banu)))
))
```

2. **Modifying the parsed tree structure based on the language:**

3. **Lemmatizing tokens**: Lemmatizing is used to extract the stem or the base form of a certain word. The lemmatized tokens of the input tokens **"dogs, troubled, apples"** would be **"dog, trouble, apple"**. Python WordNetLemmatizer [45] is used to develop the prototype. The figure 3.8 shows the code snippet of the lemmatize_tokens method.

4. **Generating fingerspelling:** The fingerspelling system is used to represent the letters of a given word. The figure 3.9 shows the code snippet of the fingerspelling method of the proposed architecture. The main goal of this method is to extract the finger-spelling characters from a given sentence. In this architecture, a text file is used to store the basic sign forms that already

exist in the system. Following is a sample of the word file that stores the available signs.

```
['0', 'axe', 'colour', 'forgive', 'tools', '1', 'bad',
'colours', 'form', 'lock', 'read', 'touch', '10',
'badminton', 'come', 'four', 'long', 'ready', 'toward',
'100', 'bag', 'lorry', 'receive', 'town', '11', 'bake',
'lose', 'reception', 'track', '12', 'communicate',
'fourteen', 'loss', 'rectangle', 'trade-equipment',
'13', 'balloon', 'communication', 'france', 'loss1',
'red', 'train', 'bandage', 'compare', 'friday',
'lotus', 'regions', 'transport' ]
```

Every signing language should maintain such files in the system. Based on the selected language by the end-user, the system will read the appropriate word file and identify whether the input word exists or not. If the word is not available, then the word is converted into the basic characters of the given language. Each character has an associated sign in every signing language.

3.3.5 Client Event Handler

As shown in the figure 3.10, client Event handler contains of three methods.

1. Speech recognizer
2. Request Handler
3. Tokenizer

Speech recognizer: Speech recognizer: This component recognizes the voice and converts it to text. Google Web Speech API[54] is used for the development of the prototype.

```
def pre_process(sentence):
    words = list(sentence.split())
    print ("\nGenerate fingerspelling...")
    f = open('words.txt', 'r')
    eligible_words = f.read()
    f.close()
    final_string = ""

    for word in words:
        if word not in eligible_words:
            for letter in word:
                final_string += " " + letter
        else:
            final_string += " " + word
    print (final_string)
    return final_string
```

Figure 3.9: Generating Fingerspelling

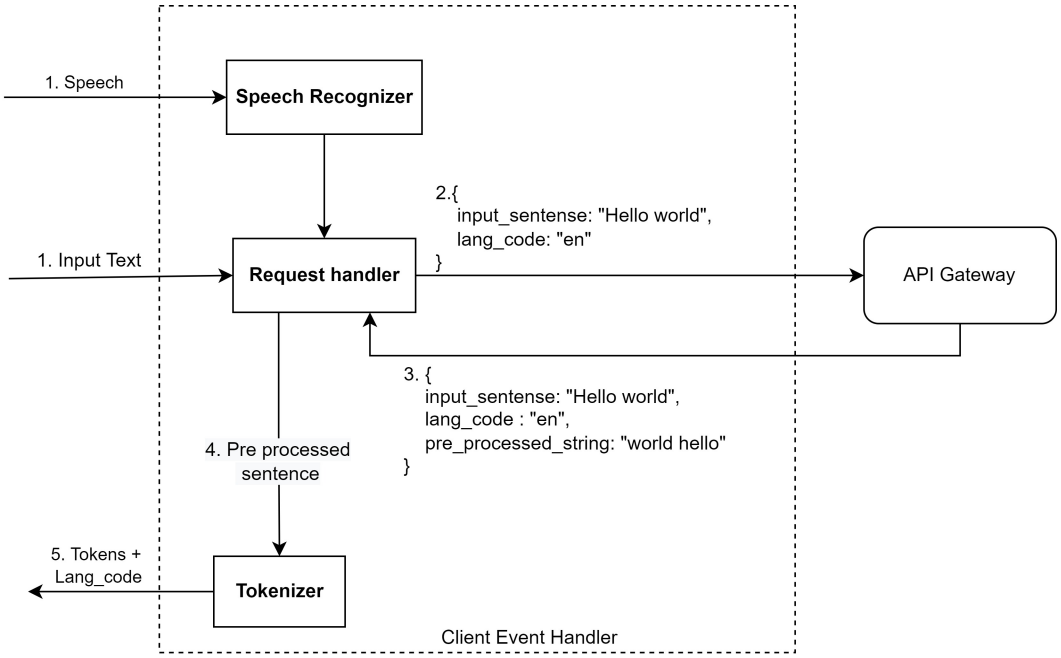


Figure 3.10: Client Event Handler

Request Handler: Request handler has been called if the speech recognition is triggered or the *'Convert'* button of *UI Client* has been clicked after inserting an input text. Once the request handle has an input text, it fetches the language code from the Client UI. Then combine the input sentence and the language code into a JSON string. Then invoke the *LanguageParserHandler* REST API along with the JSON parameter values. API gateway then responds with a preprocessed string after invoking the relevant microservices which is responsible for parsing the sentence. The response of the language parser handler is a JSON string that contains the input sentence, language code, and the output sentence (refer to the Figure 3.4 under subsection 3.3.3 to understand the output of the *LanguageParser-Handler* service). Once the Request handler receives the request, the preprocessed sentence has been extracted from the response JSON string and forwarded to the *Tokenizer* method.

Tokenizer: This method tokenizes the input sentence into tokens and forward a token array to the SiGML converter module.

3.3.6 SiGML converter

SiGML converter has two methods shown in the figure 3.11.

1. Tokens to SiGML mapper

2. Animation Player

Tokens to SiGML mapper: This method maps the tokens into relevant SiGML files. To check if a SiGML file is available for a given token, a PHP web service is invoked. The token is passed as a parameter to the API as an input. A JSON file is used to store the mapping between the token and the SiGML files. The format of the SiGML dictionary is shown in the figure 3.12. SiGML dictionary contains the SID, the name of the Token, and the mapped SiGML file.

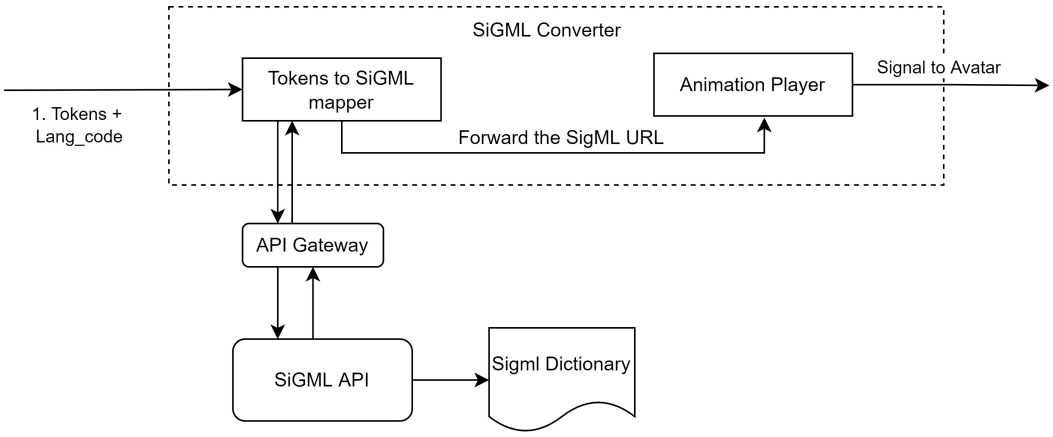


Figure 3.11: SiGML Converter


```
[
  {
    "sid": 299,
    "name": "0",
    "fileName": "0.sigml"
  },
  {
    "sid": 671,
    "name": "1",
    "fileName": "1.sigml"
  },
  {
```


Figure 3.12: SiGML Dictionary

The PHP web Service reads the JSON file and retrieves the matching SiGML file for a specific token. If no matching SigML file is available, then the token is broken into fingerspelling characters and mapped the relevant SiGML file with those identified characters. Response of the SiGML API is another JSON string. It contains a stream of all the SIGML files that need to be executed on the client side.

Animation Player: Animation player calls the Avatar library (marc.jar) to generate actual signs.

Once the Tokens to SiGML mapper has the SiGML file names that need to be sent to the Avatar, the animation player is invoked for each SiGML file. The method parameter is the path of the SiGML file.

CHAPTER IV

Evaluation of the Architecture

This chapter discusses the evaluation of text to sign language interpreter system. Architectural analysis is the process of finding important system properties using the architectural model of a system. Two major evaluation techniques are being used here. Architecture Trad-off Analysis Method (ATAM) is used to evaluate the quality attributes of the architecture. And the proposed architecture is compared with an existing project.

4.1 Quality Attributes

Before evaluating the Architecture, the quality attributes of the system should be identified. As explained in the section2.5, due to trade-offs, some quality attributes cannot be achieved together. The proposed architecture mainly focuses on the following quality attributes.

- Maintainability

Maintainability is achieved in this architecture by proposing a microservice architecture. For example, a language parser handler is a microservice which is introduced abstraction to the system. Client portion can be easily maintained without knowing the actual services available in the back end. A property file is used to maintain all the available services depending on the language that is requested by the client. New language parser services can be introduced without changing a single line of the code. It's just a matter of modifying the property file

and including all the services that are available to use. With the usage of the API gateway, another level of abstraction is introduced to the system. All the requests travel through the API gateway and the routing is handled by the API gateway itself. Additional services can be introduced without hassling the existing codes. Just need to configure the API gateway to reroute the request from the client.

Dividing features into components. Small feature components can be easily maintained without a major effort. Different technologies can be used to develop these components. Features or functionalities are carefully extracted from the system and designed as separate components. Multiple components are maintained easily compared to one single component.

Other maintainability aspects can be achieved by maintaining the versions of the code in a repository. User version controlling is provided better maintainability to the system. Also, maintaining a good coding standard with better commenting practices is a must. With the above-mentioned aspects, the following benefits can be obtained.

Components are understandable compared with systems that has not been followed any component architecture. Bugs can be identified easily with the component architecture. Individual components can be tested to identify bugs. Deployment can be easily done without affecting the other components.

- Modifiability

The modifiability aspect measures how easy to change the existing code without too much affecting the existing system and how the system accepts such changes. This quality attribute can be achieved by the microservices architecture proposed in the system. Since each microservice is processed separately, other services are not aware of any changes to be made for one service. Also, the Language Parser Handler Service is not aware of the changes that are made to a particular service. With this microservice concept, the modifiability aspect can be achieved easily.

API gateway is existed in between client UI and the REST APIs. Any changes to this module can be introduced without affecting the system. Also, the entire API gateway can be replaced by a different component with different technologies.

- Reusability

The reusable aspect is achieved with the Language parser hander service. As explained in the previous chapter, this component act as an abstraction layer to the client and the microservices. Any new language parser handler service can be introduced to the system just by changing the configuration settings in the property file. An entire new sign language will be using the existing components without changing or affecting them. The output signing language will be adjusted based on the selected value in the client UI. All the components will be reused for any sign language except the Language parser service. Language parser service is specifically developed to cater to sign language-specific requirements.

- Interoperability

This architecture was designed based on service-oriented architecture. The services that provide language-specific computation are REST APIs. A JSON string is provided as an input to the REST APIs and the output is obtained as a JSON document. With this design, these REST APIs can work with any external program that follows the same standards. APIs are not aware of the sender of the request but process it and provide the desired output in the standard way. Services can be developed using different technologies. The consumer of the services can also be developed using any technology. The communication between those two can be done using HTTP calls. With the nature of exposed REST APIs, the underline technology will not be mattered. The Only requirement is, both the application and the service should have the ability to understand the HTTP requests and responses.

- Testability

Testability refers to the level at which software artifacts assist in testing a given test condition. If the software artifact is highly testable, system troubleshooting will be easier. Since the proposed architecture supports service-oriented architecture, each REST API can be tested separately. A tool like Postman or JMeter[1] can be used for this. As the SIGML player is also a separate module, it can be tested separately.

- Performance

The performance of this proposed microservices architecture will be slightly degraded due to the multiple routes compared to a monolithic architecture. To improve the performance of the system few modifications have been done to the Language parser handler services. MongoDB NoSQL database is used to store the input sentence against the intermediate sentence which is parsed by the language parser services. If the client requests a sentence to be translated, and that sentence was already requested by the same cline or different client previously, then language parser handler services won't be called the relevant language parser service. Instead, it will be fetched the intermediate sentence directly from the NoSQL database. By doing this, the computational time for language processing will be reduced. Response time for the already parsed sentence is reduced.

A load balancer can be introduced between the client and the API gateway to improve the performance of the system if is scaled out. Load balancers can be used to redirect the request to multiple instances of the same API gateway which is connected with the same set of microservice as they scale out. With the above-mentioned changes, the performance of the system can be improved to cater to multiple requests. The availability of the system also can be improved at the same time.

[1] An Apache project that can be used for load testing.

4.2 ATAM - Architecture Trad-off Analysis method

ATAM method is used to evaluate the proposed architecture (Refer to the subsection 2.5.1 for detailed steps). For the documentation purpose, all the nine steps are grouped into the following four sections Presentation, Investigation and Analysis, Testing, and reporting.

Evaluation team: A software architect, a technical lead, a Developer, and a Project Manager are involved in this evaluation process.

The evaluation process was done mainly via virtual meetings and through emails. A teams meeting was set up to discuss the presentation stage which included presenting the ATAM, Presenting the business drivers, and the Present the architecture.

Presentation

1. **Present the ATAM**: Initially, the ATAM method has been explained to the evaluation team via a Teams meeting. All four members had a better idea about the ATAM method since they are industry experts. The purpose of the ATAM has been explained to the members during the meeting. It is to estimate the impacts of architectural decisions considering the quality attribute requirements[58]. And explained the outcomes of the process. The evaluation team had a better idea about Risks, sensitivity points, and tradeoff points which will be the potential future concerns with the architecture.

2. **Present the business drivers:** In this step, the initial system design has been presented to the evaluation team. The motivation of this research (which is described in the section 1.1), Aims and objectives (which is described in the section 1.2), and scope (which is described in the section 1.3) were presented. Next, the functional and non-functional requirement was presented in the latter part of the presentation. Following are the presented functional, nonfunctional, and quality attributes.

Functional requirements

- Convert a given text into given sign language
- Fingerspell any none-existing words.
- Able to add additional language to the system without changing the code.
- New language parser services can be added to the system.
- Can be able to add new signs.
- Can be able to check the existence of a particular SiGML file.
- Storing data for future usage

Non-functional requirements

- User-friendly interface
- Performance of the system should be high
- Should easily test the components separately.

Quality attributes

- Maintainability
- Modifiability
- Interoperability
- Testability
- Performance
- Scalability

3. **Present the architecture**: This is the last step of the presentation stage. Here, the initial architecture was presented to the members. This was done

via a series of sessions and via email iteratively until the final architecture was modeled. Industry experts in the team have reviewed the initial architecture and corrected a few issues. Finalized architecture diagram can be shown in the figure 3.1.

Investigation and Analysis

4. **Identify the architectural approaches**: In this step, the architect had reviewed the proposed architecture. The proposed architecture is designed based on microservices architecture.

   During the analysis, a suggestion came from the architect that the location transparency of the services can be achieved with service discovery. With this feature, the client can access different instances of microservices without knowing where it located. High availability of the system can be maintained with location transparency since the services can be moved or can be migrated to a different physical location without affecting the service it is being provided. Also, the portability and the maintainability quality attributes will be affected by the location transparency architectural approach.

   With this proposed microservices architecture, encapsulation can be achieved between partitions of the application. Services can be accessed only through REST API calls. With the help of the microservices architecture, the system can be scaled easily. A load balancer can be introduced between the client and the API gateway to achieving that. With this approach, the interoperability, modifiability, scalability, and availability quality attributes can be achieved.

   Language parser handler service has introduced abstraction to the system. New services can be added to the system with a configuration change to the property file associated with the Language parser handler. The modifia-

bility and the maintainability quality attributes are mainly affected by this approach.

5. **Create a quality attribute tree**: The figure 4.1 shows the created utility tree by involving all the members of the team. The quality attributes are prioritized from top to bottom.

6. **Analyze the architectural approaches:** In this step, together with the team, the generated utility tree was analyzed. Through that, the architectural approaches were explored to identify the most prioritized quality attributes. The outcome of the analysis provided risks, sensitivity points, and tradeoffs of the architecture. Following questions were raised during the session.

    - What are the architectural methods used?
    - Are there weaknesses in the selected methods?
    - What are the sensitivity points that can affect architecture?
    - What are the tradeoffs?

    During the analysis, the following quality attributes were given the highest priority.

    - Modifiability
    - Interoperability
    - Maintainability

    By analyzing the initial architecture design, an API gateway was introduced as an abstraction layer to the system between client and microservices. However, if the API gateway is failed to process the request, the system is at a risk. Also, this is a sensitivity point of the system since all the traffic to the

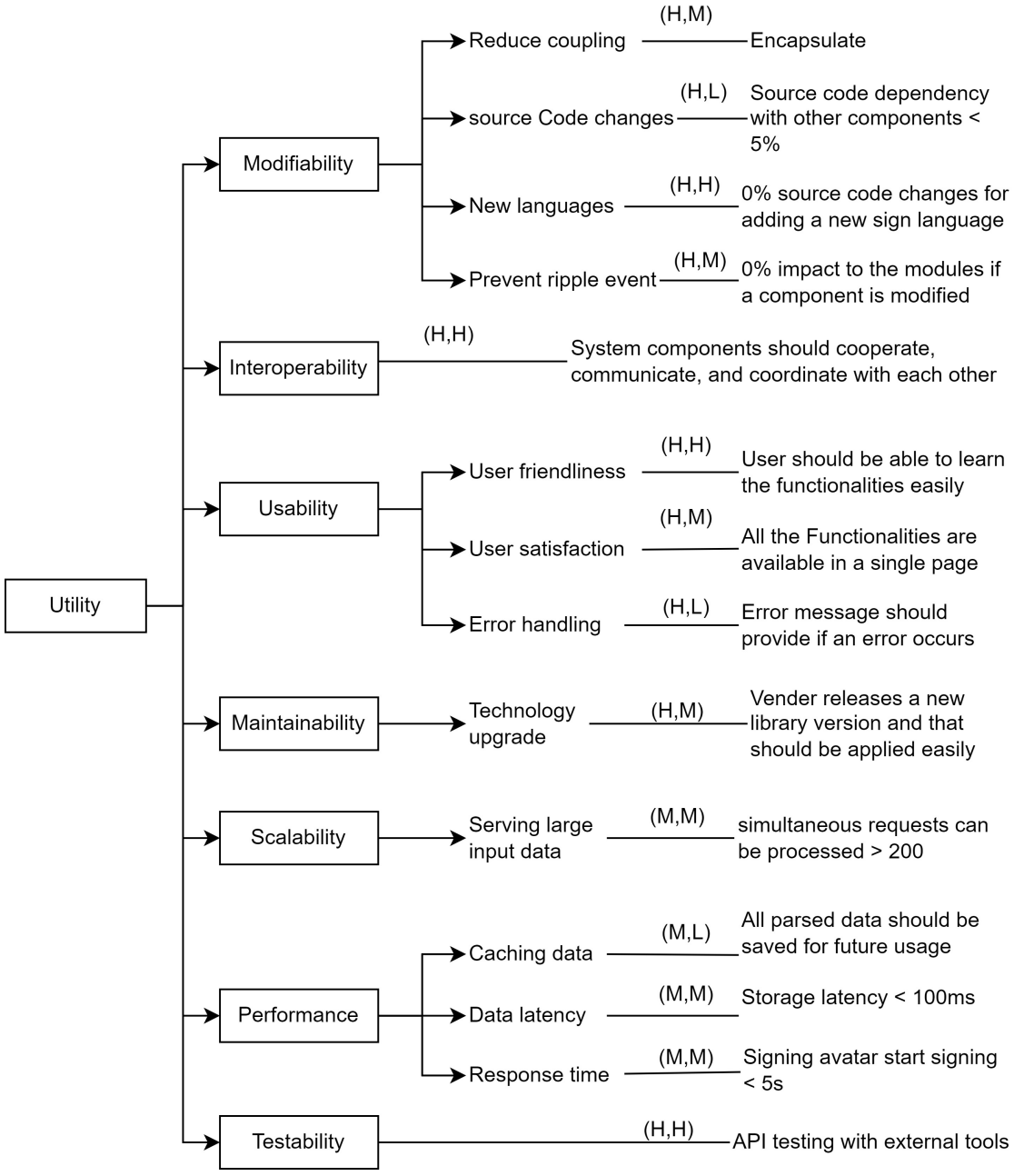


Figure 4.1: Utility tree

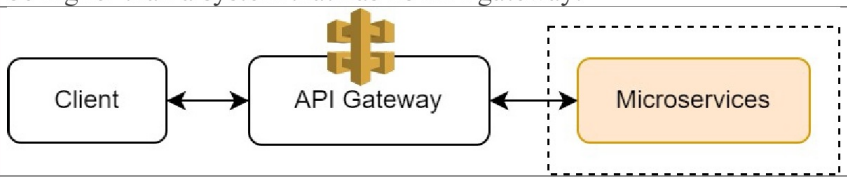
Scenario #1	Encapsulate REST API from the client			
Attribute	Modifiability, Maintainability			
Stimulus	Routing traffic			
Architecture decision	**Sensitivity**	**Risk**	**Tradeoffs**	**Non-risk**
API Gateway	S1	R1	T1	
Reasoning	**S1:** API gateway is a sensitivity point as overall system traffic is going through this component. **R1:** There is a risk if the API gateway is failed. Also, it is impacting the availability of the system. **T1:** The overall performance is impacted by the API gateway. Since the overall traffic is routed through the API gateway, the response time will be higher than a system that has no API gateway.			
Architecture diagram				

Table 4.1: Modifiability- Architecture approach description - Scenario #1

microservices are going through this point. Since the overall traffic is routed through the API gateway, the performance of the system will be degraded. This was identified as a tradeoff between modifiability and performance. The table 4.1 shows an example scenario that was taken into discussion. It shows the architecture approach description of the modifiability quality attribute.

The table 4.2 shows another scenario that was taken under the same modifiability quality attribute. The language parser handler service was introduced to the system to increase the modifiability quality attribute. This is also a sensitivity point since all the language-specific traffic is going through this service. This is the most important component in this architecture. It has provided the flexibility of adding new languages to the system without changing the code. Modifying a property file is enough to add a new language. With this component, the overall system performance may be reduced since this service acts as an extra routing point. Therefore, an extra delay is added to the throughput by the Language parser handler service. The trade-

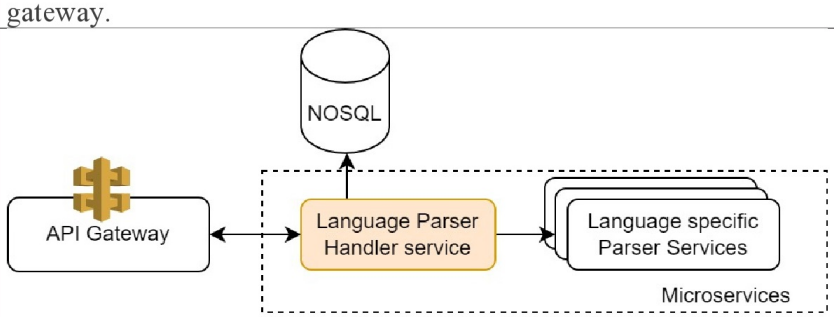
Scenario #2	0% source code changes for adding a new sign language			
Attribute	Modifiability, Maintainability			
Stimulus	Adding a new Language			
Architecture decision	**Sensitivity**	**Risk**	**Tradeoffs**	**Non-risk**
Language Parser Handler	S2		T2	✓
Reasoning	**S2:** Language Parser Handler is a sensitivity point as language parser specific system traffic is going through this component. New language ca be added to the system by modifying the property file. **T2:** The overall performance is impacted by the Language Parser Handler. Since the overall traffic is routed through the Language Parser Handler, the response time will be higher than a system that has no API gateway.			
Architecture diagram				

Table 4.2: Modifiability- Architecture approach description - Scenario #2

off between the performance and the modifiability was identified during the analysis.

Testing

7. **Brainstorm and prioritize scenarios:** During step 5, 14 scenarios were identified. After that, team members were given the chance to merge the scenarios that were related or similar. After the merging, scenarios were prioritized based on the voting mechanism. In addition to the scenarios defined in the utility tree, members were given a chance to add new scenarios based on their viewpoints. All the members were given 4^2 votes. The scenarios that

[2]

(a) The vote count per member was calculated by taking the rounded nearest integer value of 30% of the number of brainstormed scenarios [57]

#	Scenario description	Votes
2	Source code dependency with other components $< 5\%$	4
4	0% source code changes for adding a new sign language	4
5	Components should communicate/coordinate with each other	2
8	Error message should provide if an error occurs	1
13	Signing avatar start signing < 5s	2
10	simultaneous requests can be processed > 200	3

Table 4.3: Brainstorm scenarios

were suggested by team members and their votes are shown in the Table 4.3.

8. **Re-analyze the architectural approaches:** In this step, team members went through several repetitions of steps 6 and 7 to refine the output of the ATAM analysis.

4.3 System comparison with an existing application

In this section, the system's output token stream is compared to an existing system developed to translate text into Indian sign language. To compare the results of this evaluation, any text to the Sign-language translation tool can be used. However, it is the ISL project that has been chosen for this comparison. Additionally, the response time of the two systems is compared. In the development phase of the prototype, the initial sign parsing library is taken from the git project reference in the [59]. Moreover, Indian sign SIGML files have been taken from the git project referenced in [60]. The BSL SIGML example files are taken from CWASA project reference in [48].

4.3.1 Evaluation procedure

In this subsection, the evaluation procedure of the research of comparing the developed prototype with an existing application is discussed. Several mechanisms were followed for a better evaluation. The source sentences were selected from dif-

ferent sources, ensuring that they were as simple as possible. For example, some sentences were extracted from novels and newspapers randomly.

Those randomly selected sentences were provided as inputs to both systems and the output was recorded. Since the proposed prototype was developed according to the microservices architecture, the language parse handler service was tested using the Postman tool. Meantime, the existing ISL project was tested using the same input sentences and output was recorded. The table 4.4 depicts the comparison between the output token of the proposed language parser handler service and the ISL project.

Additionally, the response time of both the systems was calculated, using the Postman tool for the language parser hander service, and using the browser developer tool interface for the ISL application. The table 4.5 depicts the results of the response time of both systems.

4.3.2 Result analysis

As shown in the table 4.4, fifteen sentences were chosen as the input sentences. The output is identical for both systems as per the test. Both the systems are using the same language parser functionalities but with different architectural approaches. ISL application is a monolithic system while the proposed architecture is a microservices application. Yielding the same output by both systems prove that the proposed system has not affected the functionality of the base application. A BSL system also can be tested in the same way. The output is highly reliant on the available sign tokens. If a particular sign token is not available, the relevant fingerspelling tokens are provided by the fingerspelling generator. As shown in the table 4.4 fingerspelling tokens can be denoted in single characters.

According to the table 4.5, the response time of the ISL project is around 1700ms for a simple SVO sentence. The processing time of the language parsing for Indian Sign language is highly impacted on the response time of the ISL

#	Input sentence	Output token of the text to ISL project [59]	Output token of the Language parser handler service
1	my name is Banu	my name b a n u	my name b a n u
2	Does he play tennis	he tennis d o e s p l a y	he tennis d o e s p l a y
3	He drives to work	he d r i v e to work	he d r i v e to work
4	she likes banana	s h e b a n a n a l i k e	s h e b a n a n a l i k e
5	Mary enjoys cooking	m a r y enjoy cook	m a r y enjoy cook
6	You run to the party	you p a r t y run to	you p a r t y run to
7	boy is eating an apple	boy apple e a t i n g	boy apple e a t i n g
8	A girl teaches tamil	girl teach tamil	girl teach tamil
9	mother needs a sewing machine	mother sewing machine need	mother sewing machine need
10	They attend an important work	they important work attend	they important work attend
11	He is busy	he busy	He is busy
12	I got the certificate	i certificate g o t	i certificate g o t
13	my mother is sad	my mother sad	my mother sad
14	give me a call	me call give	me call give
15	please remind me	me please remind	me please remind

Table 4.4: Token output comparison: ISL project [59] vs Proposed prototype

application.

As shown in the table 4.5, the response time of the very first attempt of the proposed prototype is a bit higher than the response time of the ISL system. However, the response time of any repetitive attempt by any client is drastically reduced in the proposed prototype. In the proposed architecture, the average response time of the very first attempt for a given sentence is around 3900ms (Simple SVO-type sentences are only considered). However, the average response time of the n^{th} attempt excluding the first attempt is around 190ms.

Reason of this behaviors can be explained using the architectural approaches that have been used in the proposed architecture. The Response time of a certain input sentences in the first attempt is dramatically high due to the extra routing time and communication delay added from the API gateway and the Language parser handler service. The Average Response time of a certain input sentence in

#	Input sentence	Response time of ISL project [59] – in millisecond (ms)					Response time of the prototype of the proposed architecture - in millisecond					
		a_1	a_2	a_3	a_4	Avg_1	a_1	a_2	a_3	a_4	a_5	Avg_2
1	my name is Banu	1409	1776	1430	1759	1593.5	3827	337	43	348	32	190
2	Does he play tennis	3309	1387	1817	1475	1997	4151	326	22	327	22	174.25
3	He drives to work	1698	1432	1725	1470	1581.25	4191	347	26	376	26	193.75
4	she likes banana	1734	2123	1424	1700	1745.25	3852	26	375	24	365	197.5
5	Mary enjoys cooking	1749	1754	1448	1749	1675	3828	28	367	26	332	188.25
6	You run to the party	1453	1735	1413	1778	1594.75	3854	29	388	26	351	198.5
7	boy is eating an apple	1806	1522	1816	1449	1648.25	3921	27	334	30	368	189.75
8	A girl teaches Tamil	1786	1756	1400	1695	1659.25	3830	27	364	24	328	185.75
9	mother needs a sewing machine	1894	1517	1465	1794	1667.5	3872	30	373	26	352	195.25
10	They attend an important work	1483	1819	1485	1817	1651	3887	41	346	22	377	196.5
11	He is busy	1390	1712	1426	1701	1557.25	3805	24	366	29	327	186.5
12	I got the certificate	1461	1769	1460	1724	1603.5	3909	33	339	29	389	197.5
13	my mother is sad	1957	1633	1799	1847	1809	3787	24	328	27	367	186.5
14	give me a call	1436	1755	1735	2039	1741.25	3906	47	335	48	330	190
15	please remind me	1688	1738	1901	1599	1731.5	3890	27	332	22	329	177.5

* a_n: Response time of the n^{th} attempt; Avg_1: Average response time of the ISL project; Avg_2: Average response time of the 2^{nd}, 3^{rd}, $4^{th}$and 5^{th}attempt of the proposed architecture

Table 4.5: Response time comparison: ISL project [59] vs Proposed prototype

the first attempt of the proposed architecture can be shown as in the following equation.

$$ART^P_{a_1} = RT_{APIGateway} + RT_{LPH} + T_{LP} + CD$$

*$ART^P_{a_1}$: The Average Response time of a certain input sentences in the first attempt of the proposed architecture

*$RT_{APIGateway}$: Routing time of the API gateway

*RT_{LPH}: Routing time of the language parser handler service

*T_{LP}: Actual language parsing time

*CD: Communication delay

However, at the first attempt, the output token set is stored in a NoSQL database as a JSON string. With that architectural approach, from the second

attempt, for the same input sentence, the output token-set is fetched from the NoSQL database instead of going through the language parser process again. This approach is helped to reduced the response time of the system from the second attempt onwards.

$ART^{P}_{a_1} > ART_{ISL} > ART^{P}_{a_n}(n \neq 1)$

*ART_{ISL}: The Average Response time of the ISL application for a certain input sentences

*$ART^{P}_{a_n}(n \neq 1)$: The Average Response time of a certain input sentences in the n^{th} attempt of the proposed architecture

As described, The proposed system provides better performance compared to the ISL system in response time point of view from the second attempt onwards for a given sentence.

CHAPTER V

Conclusion

In this research, an architecture for text to sign language is proposed to address the following issues: Although there are numerous "Text to sign language interpreters" are available for different sign languages, researchers/students have to develop the entire prototype/application to test/demonstrate the research outcome. Assume a research student has developed a text-to-sign translation library. To evaluate the new sign language library, they must spend time and effort on developing a prototype that is totally out of the scope of their research topic. If there is an application available and its architecture supports any sign language library with minimum configuration changes, then researchers can use it as a model and save their time and effort for the actual problem they are working on. This is the main goal of this research.

In addition to that, as a tool, it can be used to fill the gap between the deaf and non-deaf communities. It is not possible to use the written form of a language to communicate with deaf society since reading and writing ability is very poor with deafness. Another advantage is that this tool can be used to translate news programs into sign languages.

The main contribution of the research is the proposed microservices architecture which has the flexibility of supporting multiple sign languages. With this proposed architecture, any sign language can be added to the system without changing the code. It's just a matter of configuring a property file associated with the component introduced in this research called the "Language parser handler" service.

This service act as an abstraction layer between the client and exposed RESTFul microservices. It is connected to a NoSQL database to store the parsed output for future usage. As proved in the evaluation section, the response time of the system is reduced with this architectural approach.

Other main components of the proposed architecture are language parser services, API gateway, Client event Handler, Sigml converter. The language processing part is done by the language parser services. These are microservices that can be added to the system by changing the configuration file. The system identifies the services by reading the property file associated with the language parser handler. API gateway act as an abstraction layer between the client and the backend services.

Under the Evaluation chapter, two main evaluation techniques were used. The architecture trade-off analysis method (ATAM) has been used to identify the Risks, sensitivity points, and trade-offs of the system. Four members were involved in the evaluation. According to the prioritization, six scenarios were identified during the ATAM evaluation. As a result of the evaluation, the Language parser handler service and the API gateway are added to the architecture to facilitate the modifiability and maintainability quality attributes. However, with these new components, the overall performance of the system is reduced, and it has been identified as a tradeoff between performance and modifiability.

The second evaluation method was to compare the developed prototype with an existing tool. ISL project [59] which was developed to translate text to Indian sign language has been used for the comparison. To compare the results of this evaluation, any text to the Sign-language translation tool can be used. However, it is the ISL project that has been chosen for this comparison. The main difference between our prototype and the ISL project is, in our architecture, the avatar processing part is offloaded to the client-side. Additionally, we proposed a microservices architecture.

To compare the two systems, 15 simple sentences were selected randomly from different sources. Those sentences were provided to both systems and compared the output. According to the results, both systems were given the same output in different formats. In our prototype, the output would be a JSON string.

Then the response time of both systems was compared. According to the results, the average response time of a certain input sentence in the first attempt of the proposed system is much higher compared to the ISL project. The reason is, there is a tradeoff between modularity and the performance of the system. The response time of the proposed system is increased with the component that has been introduced to facilitate modifiability and maintainability. To decreased the response time, the NoSQL database was added to store the preprocessed tokens. The analysis proved that the architectural decision that has been taken to store the pre-procced tokens in the NoSQL database is impacted to reduce the response time from the second attempt onwards.

Concluding that, the proposed architecture is supported multiple sign language with minimum configuration changes. The results show that overall response time is decreased with the second attempt onwards.

5.1 Discussion

The proposed architectural model is the main outcome of this research. Any sign language can be added to the system using this framework without changing a single line of code. This will be beneficial to researchers and students conducting research in the field of sign language. Assume a graduate student created a text-to-sign translation library. They should spend time and effort developing a prototype that is completely outside the scope of their research topic in order to evaluate that new sign language library. Researchers can use this framework as a model because its architecture supports any sign language library with minimal configuration changes. So that they can concentrate solely on the problem at hand. That is the

main advantage that is provided for the researchers.

Like Google Translator, this framework can be used as a platform for translating natural languages into Sign languages. This is a huge benefit in bridging the gap between the deaf community and the rest of society. There are tools for translating the text into Sign languages. Those, however, are limited to specific sign languages. This microservices architecture supports any Sign language, which is the main contribution of this research and the most significant benefit.

5.2 Future Work

To liaise with the deaf community, two-way communication is very important. This can be achieved by text to sign and sign to text translation. In this research, the proposed architecture has addressed only the text to sign language translation. As future work, the proposed architecture needs to be extended to support sign-to-text translation as well.

The proposed architecture can be improved by introducing the service discovery for microservices. As per the current design, the host and the port of the microservices are well known. Therefore, it is hard to scale up the system by adding multiple instances of the same microservice. By introducing the service discovery pattern, the number of instances of a service can be increased based on the incoming load. As a result of that architectural approach, the system can be scaled up to cater to a high volume of incoming requests.

References

[1] M. Kipp, A. Heloir, and Q. Nguyen, "Sign language avatars: Animation and comprehensibility," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6895 LNAI, pp. 113–126.

[2] "Central Fedaration of the Deaf." [Online]. Available: http://www.slcfd.lk/sign_ language.php. [Accessed: 02-January-2020].

[3] R. San-Segundo, R. Barra, R. Córdoba, L. F. D'Haro, F. Fernández, J. Ferreiros, J. M. Lucas, J. Macías-Guarasa, J. M. Montero, and J. M. Pardo, "Speech to sign language translation system for Spanish," *Speech Communication*, vol. 50, no. 11–12, pp. 1009–1020, Nov. 2008, doi: 10.1016/j.specom.2008.02.001.

[4] C. Wideman and M. Sims, "Signing avatars," *Technology & Persons with Disabilities Conference*, pp. 249–251, 1998.

[5] K. K. El-darymli, O. O. Khalifa, and H. Enemosah, "Speech to Sign Language Interpreter System (SSLIS)"

[6] W. C. Stokoe and M. Marschark, "Sign language structure: an outline of the visual communication systems of the American deaf. 1960.," *Journal of deaf studies and deaf education*, vol. 10, no. 1, pp. 3–37, Jan. 2005, doi: 10.1093/deafed/eni001.

[7] J. Segouat, "A study of sign language coarticulation," *ACM SIGACCESS Accessibility and Computing*, no. 93, pp. 31–38, Jan. 2009, doi: 10.1145/1531930.1531935.

[8] J. Sansonnet, A. Braffort, and C. Verrecchia, "Towards interactive web-based virtual signers: first step, a platform for experimentation design," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5934 LNAI, pp. 313–324, doi: 10.1007/978-3-642-12553-9_28..

[9] R. Cole, J. Beskow, J. Yang, U. Meier, A. Waibel, P. Stone, A. Davis, C. Soland, G. Fortier, T. Carmell, P. Connors, M. Macon, J. Wouters, J. de Villiers, A. Tarachow, D. Massaro, and M. Cohen, "Intelligent animated agents for interactive language training," *ACM SIGCAPH Computers and the Physically Handicapped*, no. 61, pp. 5–10, Jun. 1998, doi: 10.1145/288076.288077.

[10] T. Dasgupta, S. Shukla, S. Kumar, S. Diwakar, and A. Basu, "A Multilingual Multimedia Indian Sign Language Dictionary Tool," *IJCNLP*, 2008.

[11] X. Finite, S. Toolkit, and L.- Martin, "A Morphological Analyzer for Verbal As- pect in American Sign Language," 2003.

[12] R. Éva Sáfár and I. Marshall, "Translation of English Text to a DRS-based, Sign Language Oriented Semantic Representation," 2001

[13] "Fingerspelling in Sinhala | Found In Ceylon." [Online]. Available: http://www.foundinceylon.com/2007/04/16/fingerspelling-in-sin**Apache HBase**hala/. [Accessed: 03-Jun-2020].

[14] C. Valli and C. Lucas, Linguistics of American Sign Language: An Introduction, Third edit. Washington, United States of America: An imprint of Gallaudet University Press, 2002, p. 493.

[15] S. K. Liddell, "Grammar, Gesture, and Meaning in American Sign Language," 2003.

[16] A. Heloir, S. Gibet, F. Multon, and N. Courty, "Captured motion data processing for real time synthesis of sign language," *Gesture in Human-Computer Interaction and Simulation*, no. LNCS/LNAI 3881, pp. 168–171, 2006.

[17] American Sign Language Video Dictionary and Inflection Guide. (2000). [CD-ROM]. New York: US. National Technical Institute for the Deaf, Rochester Institute of technology. ISBN: 0-9720942-0-2.

[18] El-darymli, K. K., Khalifa, O. O., & Enemosah, H. (n.d.). "Speech to Sign Language Interpreter System (SSLIS)" in *JSMICS*, 2008.

[19] S. Liwicki and M. Everingham, "Automatic recognition of fingerspelled words in British sign language," *Proceedings of CVPR4HB'09. 2nd IEEE Workshop on CVPR for Human Communicative Behavior Analysis*, no. Miami, Florida, pp. 50–57, 2009.

[20] R. Elliott, J. R. W. Glauert, J. R. Kennaway, and I. Marshall, "The development of language processing support for the ViSiCAST project", In *Proceedings of the fourth international ACM conference on Assistive technologies (Assets '00)*. ACM, New York, NY, USA, 101-108. 2000, pp. 101–108, doi: 10.1145/354324.354349.

[21] S. Cox, M. Lincoln, and J. Tryggvason, "The development and evaluation of a speech-to-sign translation system to assist transactions," *International Journal of Human Computer Studies*, vol. 16, no. 2, pp. 141–161, 2003, doi: 10.1207/S15327590IJHC1602_02.

[22] "Fingerspelling in Sinhala | Found In Ceylon." [Online]. Available: http://www.foundinceylon.com/2007/04/16/ fingerspelling-in-sinhala/. [Accessed: 31-Jul-2019].

[23] C. Valli and C. Lucas, "Linguistics Of American Sign Language", Third edit. Washington, United States of America: An imprint of Gallaudet University Press, 2002, p. 493.

[24] “Sri Lankan Sign Language.” [Online]. Available: https://www.rohanaspecialschool.org/sri-lankan-sign-language-dictionary/. [Accessed: 13-Jan-2020]

[25] Gouri Sankar Mishra, Parmanand Asteya, Pooja, “English text to Indian Sign Language Machine Translation: A Rule Based Method,” *International Journal of Innovative Technology and Exploring Engineering Special Issue*, vol. 8, no. 10S, pp. 460–467, May 2019, doi: 10.35940/ijitee.J1084.08810S19.

[26] Lalitha Natraj, Sujala D. Shetty, “A Translation System That Converts English Text to American Sign Language Enhanced with Deep Learning Modules”,*International Journal of Innovative Technology and Exploring Engineering (IJITEE)* ISSN: 2278-3075, Volume-8 Issue-12, October 2019, doi: 10.35940/ijitee.L3781.1081219.

[27] Taner Arsan and Oguz Ülgen, “Sign language converter”, *International Journal of Computer Science & Engineering Survey (IJCSES)*, vol. 6, no. 4, pp. 39–51, Aug. 2015, doi: 10.5121/ijcses.2015.6403.

[28] É. Sáfár & I. Marshall, “The architecture of an English-text-to- Sign-Languages translation system”, *Proceedings of Recent Advances in Natural Language Processing (RANLP)*, 2001, pp. 223-228

[29] J. A. Bangham, S. J. Cox, R. Elliot, J. R. W. Glauert, I. Marshall, S. Rankov, & M. Wells, “Virtual signing: Capture, animation, storage and transmission - An overview of the ViSiCAST project”, *IEEE Seminar on Speech and language processing for disabled and elderly people*, Apr. 2000, no. 25, pp. 23–29, doi: 10.1049/ic:20000136.

[30] “NLP Guide: Identifying Part of Speech Tags using Conditional Random Fields.” [Online]. Available: https://medium.com/analytics-vidhya/pos-tagging-using-conditional-random-fields-92077e5eaa31. [Accessed: 28-Feb-2020].

[31] M. Punchimudiyanse and R. G. N. Meegama, "3D signing avatar for Sinhala Sign language," 2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS), 2015, pp. 290-295, doi: 10.1109/ICIINFS.2015.7399026.

[32] “JASigning - Virtual Humans.” [Online]. Available: https://vh.cmp.uea.ac.uk/index.php/JASigning. [Accessed: 25-Feb-2022].

[33] “SiGML Signing App New Gui: vhg.2021.” [Online]. Available: https://vhg.cmp.uea.ac.uk/tech/jas/vhg2021/SiGML-Player-gui.html. [Accessed: 26-Feb-2022].

[34] “Welcome to Flask — Flask Documentation (2.0.x).” [Online]. Available: https://flask.palletsprojects.com/en/2.0.x/. [Accessed: 26-Feb-2022].

[35] M. Grinberg, “Flask Web Developement,” O’Reilly, no. 1, pp. 1–5, 2014.

[36] M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, I. F. Rochimansyah, and A. Rokhim, "Design an MVC Model using Python for Flask Framework Development," IES 2019 - Int. Electron. Symp. Role Techno-Intelligence Creat. an Open Energy Syst. Towar. Energy Democr. Proc., pp. 214–219, Sep. 2019.

[37] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter, "NoSQL database systems: a survey and decision guidance," Comput. Sci. - Res. Dev., vol. 32, no. 3–4, pp. 353–365, Jul. 2017.

[38] P. J. Sadalage and M. Fowler, NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, Upper Saddle River, 2013.

[39] "Apache Cassandra | Apache Cassandra Documentation." [Online]. Available: https://cassandra.apache.org/_/index.html. [Accessed: 27-Feb-2022].

[40] M. N. Vora, "Hadoop-HBase for large-scale data," Proc. 2011 Int. Conf. Comput. Sci. Netw. Technol. ICCSNT 2011, vol. 1, pp. 601–605, 2011.

[41] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," Proc. - 2011 6th Int. Conf. Pervasive Comput. Appl. ICPCA 2011, pp. 363–366, 2011

[42] "mongoengine." [Online]. Available: http://mongoengine.org/. [Accessed: 01-Mar-2022].

[43] Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In LREC 2006.

[44] "NLTK :: Natural Language Toolkit." [Online]. Available: https://www.nltk.org/. [Accessed: 01-Mar-2022].

[45] "NLTK :: nltk.stem.wordnet." [Online]. Available: https://www.nltk.org/_modules/nltk/stem/wordnet.html. [Accessed: 01-Mar-2022].

[46] "CWA Signing Avatars - Virtual Humans." [Online]. Available: https://vh.cmp.uea.ac.uk/index.php/CWA_Signing_Avatars. [Accessed: 02-Mar-2022].

[47] "Configuring CWASA for HTML5 web pages - Virtual Humans." [Online]. Available: https://vh.cmp.uea.ac.uk/index.php/Configuring_CWASA_for_HTML5_web_pages. [Accessed: 02-Mar-2022].

[48] "CWASA Local Installation - Virtual Humans." [Online]. Available: https://vh.cmp.uea.ac.uk/index.php/CWASA_Local_Installation. [Accessed: 02-Mar-2022].

[49] "Animgen-Virtual Humans." [Online]. Available: https://vh.cmp.uea.ac.uk/index.php/Animgen. [Accessed: 03-Mar-2022].

[50] "SiGML-Virtual Humans." [Online]. Available: https://vh.cmp.uea.ac.uk/index.php/SiGML. [Accessed: 03-Mar-2022].

[51] Hanke, T. (2004), "HamNoSys - representing sign language data in language resources and language processing contexts." In: Streiter, Oliver, Vettori, Chiara (eds): LREC 2004, Workshop proceedings : Representation and processing of sign languages. Paris : ELRA, 2004, - pp. 1-6.

[52] "SiGML Tools - Virtual Humans." [Online]. Available: https://vh.cmp.uea.ac.uk/index.php/SiGML_Tools. [Accessed: 03-Mar-2022].

[53] "Python Release Python 3.10.2 | Python.org." [Online]. Available: https://www.python.org/downloads/release/python-3102/. [Accessed: 04-Mar-2022].

[54] "Voice Driven Web Apps: Introduction to the Web Speech API | Google Developers." [Online]. Available: https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API. [Accessed: 05-Mar-2022].

[55] M. Narayanan and G. Shreelekhya, "Methods for evaluating software architecture-A survey," Artic. Int. J. Pharm. Technol., vol. 8, no. 4, pp. 25720–25733, 2016.

[56] J. K. Bergey, M. J. Fisher, and L. G. Jones, "Use of the Architecture Tradeoff Analysis MethodSM (ATAMSM) in Source Selection of Software-Intensive Systems," *CARNEGIE-MELLON UNIV PITTSBURGH PA Softw. Eng.* INST, 2002.

[57] M. Barbacci, A. Lattanze, L. Northrop, en W. Wood, "Using the Architecture Tradeoff Analysis MethodSM (ATAMSM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study", bl 32, 07 2003.

[58] R. Kazman, M. Klein, en P. Clements, "ATAM: Method for architecture evaluation", Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2000.

[59] "aanchal1308/Sign-Language-Translation: Sign Language Translator for Speech and Hearing Impaired." [Online]. Available: https://github.com/aanchal1308/Sign-Language-Translation. [Accessed: 20-Mar-2021].

[60] "TextToSignConversion/textToVideo-master/SignFiles at master · puneet1024/TextToSignConversion." [Online]. Available: https://github.com/puneet1024/TextToSignConversion/tree/master/textToVideo-master/SignFiles. [Accessed: 20-Mar-2021].

[61] "3.3.2 – Analyzing and Evaluating an Architecture - Architecture in Practice | Coursera." [Online]. Available: https://www.coursera.org/lecture/software-architecture/3-3-2-analyzing-and-evaluating-an-architecture-uEtkN. [Accessed: 24-Mar-2022].