B.E. PROJECT REPORT ON

**DISABILITY GUIDE (DIAD)**

**ANDROID APP**

SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS OF AWARD OF B.E. (COMPUTER ENGINEERING)DEGREE OF UNIVERSITY OF DELHI

**SUBMITTED BY:**

Vibhor Bijoy (384/CO/14)

**GUIDED BY:**

Dr. M.P.S. Bhatia

( Professor, Division of Computer Engineering)



DIVISION OF COMPUTER ENGINEERING

NETAJI SUBHS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF DELHI

2018



**CERTIFICATE**

The project titled **“Disability Guide (DIAD)”** by **Vibhor Bijoy(384/CO/14)**is a record of bonafide work carried out by me, in the Division of Computer Engineering, Netaji Subhas Institute of Technology, New Delhi, under the supervision and guidance of **Dr. M.P.S. Bhatia** in partial fulfilment of requirement for the award of the degree of Bachelor of Engineering in Computer Engineering, University of Delhi in the academic year 2017-2018.

**Dr. M.P.S. Bhatia**

Division of Computer Engineering

Netaji Subhas Institute of Technology

New Delhi

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Candidates’ Declaration**

This is to certify that the work which is being hereby presented by me in this project titled “Disability Guide ” in Partial fulfilment of the award of the Bachelor of Engineering submitted at the Department of Computer Engineering, Netaji Subhas Institute of Technology Delhi, is a genuine account of my work carried out during the period from January 2018 to May 2018 under the guidance of Dr M.P.S. Bhatia, Department of Computer Engineering, Netaji Subhas Institute of Technology Delhi. The matter embodied in the project report to the best of my knowledge has not been submitted for the award of any other degree elsewhere.

**Vibhor Bijoy**

**Date: \_\_\_\_\_\_\_\_\_\_**

This is to certify that the above declaration by the student is true to the best of my

knowledge.

**Dr. M.P.S. Bhatia**

**Date: \_\_\_\_\_\_\_\_\_\_**

**Acknowledgment**

This project’s success is directly attributed to the assistance and support from all of those individuals involved in this five month period.

I would like to express our sincere gratitude towards my mentor **Dr. M.P.S. Bhatia**, Professor, Computer Engineering Department, Netaji Subhas Institute of Technology, Delhi under whose supervision I completed my work. His invaluable suggestions, enlightening comments and constructive criticism always kept my spirits up during my work. He was always available to help whenever I faced any problems.

My experience in working under the supervision of Prof. Bhatia has been great and fruitful. I learnt a lot in this process. The knowledge, practical and theoretical, that I have gained through this project will help me in my future endeavours in the field. I also learnt the importance of working effectively and efficiently.

I am also grateful to my friends who have provided critical feedback and support whenever required.

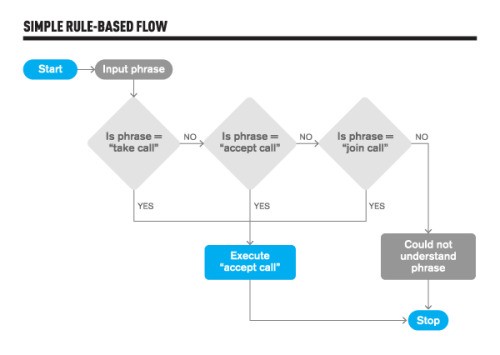
**Abstract**

The aim of this thesis is to build an android app to explore the scope of physically handicapped people under gazette notification so that they can get better opportunity in life as compared to normal people. The application is based on “Rule Based Systems or Rule based computing”. **Rule-based systems**  are used as a way to store and manipulate knowledge to interpret information in a useful way.

Rule Learning is a rule based machine learning method for discovering interesting relationship between variables and large databases . The defining characteristic of a rule based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system.

Some of the rule based machine learning approaches include :

* Learning Classifier System
* Associate Rule Learning
* Artificial immune systems



**Index**

1. **CHAPTER ONE: INTRODUCTION…………….1**
   1. MOTIVATION…………………………………….1
   2. PROBLEM STATEMENT…………………………1
   3. OBJECTIVE………………………………………….1
2. **CHAPTER TWO: WHAT IS RULE BASED LEARNING………….2**
   1. INTRODUCTION **……………….4**
   2. APPROACHES OF RULE BASED MACHINE LEARNING …………4
3. **CHAPTER THREE: NATURAL LANGUAGE PROCESSING**
   1. INTRODUCTION………….6
   2. TYPES OF NLP……….6
   3. APPLICATIONS……….8
4. **CHAPTER FOUR: EXPERIMENTATION AND RESULTS**
   1. PROCEDURE…………..10
   2. SOFTWARE SPECIFICATION …......10
   3. RESULTS………10
5. **CHAPTER FIVE : FUTURE WORK AND CONCLUSIONS**
   1. FUTURE WORK……….12
   2. CONCLUSIONS……..12
   3. REFERENCES……..12

**List of Images**

**S. No. Topic Page No.**

1. Welcome 8

2. Introduction 12

3. Flow Chart of Proposed Solution 17

4. Details 25

5. Schemes for handicapped Employees 27

6. Jobs and Education for handicapped 29

**I.**

**Introduction**

**Chapter 1. Introduction**

In computer science, **rule-based systems** are used as a way to store and manipulate knowledge to interpret information in a useful way. They are often used in artificial intelligence applications and research. A typical rule-based system has four basic components: A list of rules or **rule base**, which is a specific type of knowledge base.

* An inference engine or semantic reasoner, which infers information or takes action based on the interaction of input and the rule base. The interpreter executes a production system program by performing the following match-resolve-act cycle:
* Match: In this first phase, the left-hand sides of all productions are matched against the contents of working memory. As a result a conflict set is obtained, which consists of instantiations of all satisfied productions. An instantiation of a production is an ordered list of working memory elements that satisfies the left-hand side of the production.
* Conflict-Resolution: In this second phase, one of the production instantiations in the conflict set is chosen for execution. If no productions are satisfied, the interpreter halts.
* Act: In this third phase, the actions of the production selected in the conflict-resolution phase are executed. These actions may change the contents of working memory. At the end of this phase, execution returns to the first phase.
* Temporary working memory.
* A user interface or other connection to the outside world through which input and output signals are received and sent.

**Motivation**

To understand what is Rule Based Machine Learning and its approaches.

**Problem Statement**

The problem we focus on is :

*“Development of an android application to explore the scope of physically challenged people under gazette notification.”*

**Organisation of the Thesis**

The remaining thesis is organized as follows:

Chapter 2 carries out the literature review concerning the previous work done present in detail, i.e. about rule based systems..

Chapter 3 discusses about the concept of NLP(Natural Language Processing) that plays a crucial role in this project.

Chapter 4 discusses the demonstration of the project , software specifications and results.

Chapter 5 summarizes the thesis with conclusions and avenues of future work.

**II.**

**Rule Based**

**Learning**

**Chapter 2. Rule Based Learning**

**2.1 Introduction**

**Rule-based machine learning** (RBML) is a term in computer science intended to encompass any machine learning method that identifies, learns, or evolves 'rules' to store, manipulate or apply. The defining characteristic of a rule-based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learners that commonly identify a singular model that can be universally applied to any instance in order to make a prediction.

Rule-based machine learning approaches include learning classifier systems, association rule learning, artificial immune systems,[6] and any other method that relies on a set of rules, each covering contextual knowledge.

While rule-based machine learning is conceptually a type of rule-based system, it is distinct from traditional rule-based systems, which are often hand-crafted, and other rule-based decision makers. This is because rule-based machine learning applies some form of learning algorithm to automatically identify useful rules, rather than a human needing to apply prior domain knowledge to manually construct rules and curate a rule set.

2.2 **Approaches of RML**

**1. Learning Classifier System**

**Learning classifier systems**, or **LCS**, are a paradigm of rule-based machine learning methods that combine a discovery component (e.g. typically a genetic algorithm) with a learning component (performing either supervised learning, reinforcement learning, or unsupervised learning). Learning classifier systems seek to identify a set of context-dependent rules that collectively store and apply knowledge in a piecewise manner in order to make predictions (e.g. behavior modeling, classification, data mining, regression, function approximation, or game strategy). This approach allows complex solution spaces to be broken up into smaller, simpler parts.

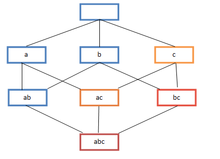
The founding concepts behind learning classifier systems came from attempts to model complex adaptive systems.

2. **Association Rule Learning**

**Association rule learning** is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness. Based on the concept of strong rules, Rakesh Agrawal, Tomasz Imieliński and Arun Swami[2] introduced association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale(POS) systems in supermarkets. For example, the rule {\displaystyle \{\mathrm {onions,potatoes} \}\Rightarrow \{\mathrm {burger} \}} found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements.

In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection, continuous production, and bioinformatics. In contrast with sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

Process

[](https://en.wikipedia.org/wiki/File:FrequentItems.png)

Frequent itemset lattice, where the color of the box indicates how many transactions contain the combination of items. Note that lower levels of the lattice can contain at most the minimum number of their parents' items; e.g. {ac} can have only at most {\displaystyle min(a,c)} items. This is called the *downward-closure property*.

Association rules are usually required to satisfy a user-specified minimum support and a user-specified minimum confidence at the same time. Association rule generation is usually split up into two separate steps:

1. A minimum support threshold is applied to find all *frequent itemsets* in a database.
2. A minimum confidence constraint is applied to these frequent itemsets in order to form rules.

While the second step is straightforward, the first step needs more attention.

Finding all frequent itemsets in a database is difficult since it involves searching all possible itemsets (item combinations). The set of possible itemsets is the power set over {\displaystyle I} and has size {\displaystyle 2^{n}-1} (excluding the empty set which is not a valid itemset). Although the size of the power-set grows exponentially in the number of items {\displaystyle n} in {\displaystyle I}, efficient search is possible using the ***downward-closure property*** of support (also called *anti-monotonicity*) which guarantees that for a frequent itemset, all its subsets are also frequent and thus no infrequent itemset can be a subset of a frequent itemset. Exploiting this property, efficient algorithms (e.g., Apriori and Eclat) can find all frequent itemsets.

1. **Artificial Immune System**

The field of Artificial Immune Systems (AIS) is concerned with abstracting the structure and function of the immune system to computational systems, and investigating the application of these systems towards solving computational problems from mathematics, engineering, and information technology. AIS is a sub-field of Biologically-inspired computing, and Natural computation, with interests in Machine Learning and belonging to the broader field of Artificial Intelligence.

Artificial Immune Systems (AIS) are adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving.

AIS is distinct from computational immunology and theoretical biology that are concerned with simulating immunology using computational and mathematical models towards better understanding the immune system, although such models initiated the field of AIS and continue to provide a fertile ground for inspiration. Finally, the field of AIS is not concerned with the investigation of the immune system as a substrate for computation, unlike other fields such as DNA computing.

AIS emerged in the mid 1980s with articles authored by Farmer, Packard and Perelson (1986) and Bersini and Varela (1990) on immune networks. However, it was only in the mid 1990s that AIS became a field in its own right. Forrest *et al.* (on negative selection) and Kephart *et al.*[2] published their first papers on AIS in 1994, and Dasgupta conducted extensive studies on Negative Selection Algorithms. Hunt and Cooke started the works on Immune Network models in 1995; Timmis and Neal continued this work and made some improvements. De Castro & Von Zuben's and Nicosia & Cutello's work (on clonal selection) became notable in 2002. The first book on Artificial Immune Systems was edited by Dasgupta in 1999.

Currently, new ideas along AIS lines, such as danger theory and algorithms inspired by the innate immune system, are also being explored. Although some believe that these new ideas do not yet offer any truly 'new' abstract, over and above existing AIS algorithms. This, however, is hotly debated, and the debate provides one of the main driving forces for AIS development at the moment. Other recent developments involve the exploration of degeneracy in AIS models, which is motivated by its hypothesized role in open ended learning and evolution.

Originally AIS set out to find efficient abstractions of processes found in the immune system but, more recently, it is becoming interested in modelling the biological processes and in applying immune algorithms to bioinformatics problems.

In 2008, Dasgupta and Nino  published a textbook on Immunological Computation which presents a compendium of up-to-date work related to immunity-based techniques and describes a wide variety of applications. The common techniques are inspired by specific immunological theories that explain the function and behavior of the mammalian adaptive immune system.

* Clonal Selection Algorithm: A class of algorithms inspired by the clonal selection theory of acquired immunity that explains how B and T lymphocytes improve their response to antigens over time called affinity maturation. These algorithms focus on the Darwinianattributes of the theory where selection is inspired by the affinity of antigen-antibody interactions, reproduction is inspired by cell division, and variation is inspired by somatic hypermutation. Clonal selection algorithms are most commonly applied to optimization and pattern recognition domains, some of which resemble parallel hill climbing and the genetic algorithm without the recombination operator.
* Negative Selection Algorithm: Inspired by the positive and negative selection processes that occur during the maturation of T cells in the thymus called T cell tolerance. Negative selection refers to the identification and deletion (apoptosis) of self-reacting cells, that is T cells that may select for and attack self tissues. This class of algorithms are typically used for classification and pattern recognition problem domains where the problem space is modeled in the complement of available knowledge. For example, in the case of an anomaly detection domain the algorithm prepares a set of exemplar pattern detectors trained on normal (non-anomalous) patterns that model and detect unseen or anomalous patterns.
* Immune Network Algorithms: Algorithms inspired by the idiotypic network theory proposed by Niels Kaj Jerne that describes the regulation of the immune system by anti-idiotypic antibodies (antibodies that select for other antibodies). This class of algorithms focus on the network graph structures involved where antibodies (or antibody producing cells) represent the nodes and the training algorithm involves growing or pruning edges between the nodes based on affinity (similarity in the problems representation space). Immune network algorithms have been used in clustering, data visualization, control, and optimization domains, and share properties with artificial neural networks.
* Dendritic Cell Algorithms: The Dendritic Cell Algorithm (DCA) is an example of an immune inspired algorithm developed using a multi-scale approach. This algorithm is based on an abstract model of dendritic cells (DCs). The DCA is abstracted and implemented through a process of examining and modeling various aspects of DC function, from the molecular networks present within the cell to the behaviour exhibited by a population of cells as a whole. Within the DCA information is granulated at different layers, achieved through multi-scale processing.

**III.**

**Natural**

**Language**

**Processing**

**Chapter 3. Natural Language Processing**

**3.1 Overview**

**Natural-language processing** (**NLP**) is an area of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to fruitfully process large amounts of natural language data.

Challenges in natural-language processing frequently involve speech recognition, natural-language understanding, and natural-language generation.

Challenges in natural-language processing frequently involve speech recognition, natural-language understanding, and natural-language generation.

**History**

The history of natural-language processing generally started in the 1950s, although work can be found from earlier periods. In 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence.

The Georgetown experiment in 1954 involved fully automatic translation of more than sixty Russian sentences into English. The authors claimed that within three or five years, machine translation would be a solved problem.[2] However, real progress was much slower, and after the ALPAC report in 1966, which found that ten-year-long research had failed to fulfill the expectations, funding for machine translation was dramatically reduced. Little further research in machine translation was conducted until the late 1980s, when the first statistical machine translation systems were developed.

Some notably successful natural-language processing systems developed in the 1960s were SHRDLU, a natural-language system working in restricted "blocks worlds" with restricted vocabularies, and ELIZA, a simulation of a Rogerian psychotherapist, written by Joseph Weizenbaum between 1964 and 1966. Using almost no information about human thought or emotion, ELIZA sometimes provided a startlingly human-like interaction. When the "patient" exceeded the very small knowledge base, ELIZA might provide a generic response, for example, responding to "My head hurts" with "Why do you say your head hurts?".

During the 1970s, many programmers began to write "conceptual ontologies", which structured real-world information into computer-understandable data. Examples are MARGIE (Schank, 1975), SAM (Cullingford, 1978), PAM (Wilensky, 1978), TaleSpin (Meehan, 1976), QUALM (Lehnert, 1977), Politics (Carbonell, 1979), and Plot Units (Lehnert 1981). During this time, many chatterbots were written including PARRY, Racter, and Jabberwacky.

Up to the 1980s, most natural-language processing systems were based on complex sets of hand-written rules. Starting in the late 1980s, however, there was a revolution in natural-language processing with the introduction of machine learning algorithms for language processing. This was due to both the steady increase in computational power (see Moore's law) and the gradual lessening of the dominance of Chomskyan theories of linguistics (e.g. transformational grammar), whose theoretical underpinnings discouraged the sort of corpus linguistics that underlies the machine-learning approach to language processing.[3] Some of the earliest-used machine learning algorithms, such as decision trees, produced systems of hard if-then rules similar to existing hand-written rules. However, part-of-speech tagging introduced the use of hidden Markov models to natural-language processing, and increasingly, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to the features making up the input data. The cache language models upon which many speech recognition systems now rely are examples of such statistical models. Such models are generally more robust when given unfamiliar input, especially input that contains errors (as is very common for real-world data), and produce more reliable results when integrated into a larger system comprising multiple subtasks.

Many of the notable early successes occurred in the field of machine translation, due especially to work at IBM Research, where successively more complicated statistical models were developed. These systems were able to take advantage of existing multilingual textual corpora that had been produced by the Parliament of Canada and the European Union as a result of laws calling for the translation of all governmental proceedings into all official languages of the corresponding systems of government. However, most other systems depended on corpora specifically developed for the tasks implemented by these systems, which was (and often continues to be) a major limitation in the success of these systems. As a result, a great deal of research has gone into methods of more effectively learning from limited amounts of data.

**3.2 Statistical Natural Language Processing**

Since the so-called "statistical revolution" in the late 1980s and mid 1990s, much natural-language processing research has relied heavily on machine learning.

Formerly, many language-processing tasks typically involved the direct hand coding of rules,[11][12] which is not in general robust to natural-language variation. The machine-learning paradigm calls instead for using statistical inference to automatically learn such rules through the analysis of large *corpora* of typical real-world examples (a *corpus* (plural, "corpora") is a set of documents, possibly with human or computer annotations).

Many different classes of machine learning algorithms have been applied to natural-language processing tasks. These algorithms take as input a large set of "features" that are generated from the input data. Some of the earliest-used algorithms, such as decision trees, produced systems of hard if-then rules similar to the systems of hand-written rules that were then common. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system.

Systems based on machine-learning algorithms have many advantages over hand-produced rules:

* The learning procedures used during machine learning automatically focus on the most common cases, whereas when writing rules by hand it is often not at all obvious where the effort should be directed.
* Automatic learning procedures can make use of statistical inference algorithms to produce models that are robust to unfamiliar input (e.g. containing words or structures that have not been seen before) and to erroneous input (e.g. with misspelled words or words accidentally omitted). Generally, handling such input gracefully with hand-written rules—or more generally, creating systems of hand-written rules that make soft decisions—is extremely difficult, error-prone and time-consuming.
* Systems based on automatically learning the rules can be made more accurate simply by supplying more input data. However, systems based on hand-written rules can only be made more accurate by increasing the complexity of the rules, which is a much more difficult task. In particular, there is a limit to the complexity of systems based on hand-crafted rules, beyond which the systems become more and more unmanageable. However, creating more data to input to machine-learning systems simply requires a corresponding increase in the number of man-hours worked, generally without significant increases in the complexity of the annotation process.

**3.3 Major Tasks**

1. **Parsing**

 Determine the parse tree (grammatical analysis) of a given sentence. The grammar for natural languages is ambiguous and typical sentences have multiple possible analyses. In fact, perhaps surprisingly, for a typical sentence there may be thousands of potential parses (most of which will seem completely nonsensical to a human). There are two primary types of parsing, Dependency Parsing and Constituency Parsing. Dependency Parsing focuses on the relationships between words in a sentence (marking things like Primary Objects and predicates), whereas Constituency Parsing focuses on building out the Parse Tree using a Probabilistic Context-Free Grammar (PCFG).

**Part of Speech Tagging**

Given a sentence, determine the part of speech for each word. Many words, especially common ones, can serve as multiple parts of speech. For example, "book" can be a noun ("the book on the table") or verb ("to book a flight"); "set" can be a noun, verb or adjective; and "out" can be any of at least five different parts of speech. Some languages have more such ambiguity than others. Languages with little inflectional morphology, such as English, are particularly prone to such ambiguity. Chinese is prone to such ambiguity because it is a tonal language during verbalization. Such inflection is not readily conveyed via the entities employed within the orthography to convey intended meaning.

1. **Natural Language Generation**

**Natural language generation** (**NLG**) is the natural language processing task of generating natural language from a machine representation system such as a knowledge base or a logical form. Psycholinguists prefer the term language production when such formal representations are interpreted as models for mental representations.

It could be said an NLG system is like a translator that converts data into a natural language representation. However, the methods to produce the final language are different from those of a compiler due to the inherent expressivity of natural languages. NLG has existed for a long time but commercial NLG technology has only recently become widely available.

NLG may be viewed as the opposite of natural language understanding: whereas in natural language understanding the system needs to disambiguate the input sentence to produce the machine representation language, in NLG the system needs to make decisions about how to put a concept into words.

A simple example is systems that generate form letters. These do not typically involve grammar rules, but may generate a letter to a consumer, e.g. stating that a credit card spending limit was reached. To put it another way, simple systems use a template not unlike a Word document mail merge, but more complex NLG systems dynamically create text. As in other areas of natural language processing, this can be done using either explicit models of language (e.g., grammars) and the domain, or using statistical models derived by analysing human-written texts.

1. **Natural Language Interpretation**

**Natural language understanding** (**NLU**) or **natural language interpretation** (**NLI**) is a subtopic of natural language processing in artificial intelligence that deals with machine reading comprehension. Natural language understanding is considered an AI-hard problem.

There is considerable commercial interest in the field because of its application to news-gathering, text categorization, voice-activation, archiving, and large-scale content-analysis.

The program STUDENT, written in 1964 by Daniel Bobrow for his PhD dissertation at MIT is one of the earliest known attempts at natural language understanding by a computer. Eight years after John McCarthy coined the term artificial intelligence, Bobrow's dissertation (titled *Natural Language Input for a Computer Problem Solving System*) showed how a computer can understand simple natural language input to solve algebra word problems.

A year later, in 1965, Joseph Weizenbaum at MIT wrote ELIZA, an interactive program that carried on a dialogue in English on any topic, the most popular being psychotherapy. ELIZA worked by simple parsing and substitution of key words into canned phrases and Weizenbaum sidestepped the problem of giving the program a database of real-world knowledge or a rich lexicon. Yet ELIZA gained surprising popularity as a toy project and can be seen as a very early precursor to current commercial systems such as those used by Ask.com.

In 1969 Roger Schank at Stanford University introduced the conceptual dependency theory for natural language understanding. This model, partially influenced by the work of Sydney Lamb, was extensively used by Schank's students at Yale University, such as Robert Wilensky, Wendy Lehnert, and Janet Kolodner.

In 1970, William A. Woods introduced the augmented transition network (ATN) to represent natural language input.Instead of *phrase structure rules* ATNs used an equivalent set of finite state automata that were called recursively. ATNs and their more general format called "generalized ATNs" continued to be used for a number of years.

In 1971 Terry Winograd finished writing SHRDLU for his PhD thesis at MIT. SHRDLU could understand simple English sentences in a restricted world of children's blocks to direct a robotic arm to move items. The successful demonstration of SHRDLU provided significant momentum for continued research in the field. Winograd continued to be a major influence in the field with the publication of his book *Language as a Cognitive Process*. At Stanford, Winograd would later be the adviser for Larry Page, who co-founded Google.

In the 1970s and 1980s the natural language processing group at SRI International continued research and development in the field. A number of commercial efforts based on the research were undertaken, *e.g.*, in 1982 Gary Hendrix formed Symantec Corporationoriginally as a company for developing a natural language interface for database queries on personal computers. However, with the advent of mouse driven, graphic user interfaces Symantec changed direction. A number of other commercial efforts were started around the same time, *e.g.*, Larry R. Harris at the Artificial Intelligence Corporation and Roger Schank and his students at Cognitive Systems corp. In 1983, Michael Dyer developed the BORIS system at Yale which bore similarities to the work of Roger Schank and W. G. Lehnart.

The third millennium saw the introduction of systems using machine learning for text classification, such as the IBM Watson. However, this is not natural language understanding. According to John Searle, Watson did not even understand the questions.

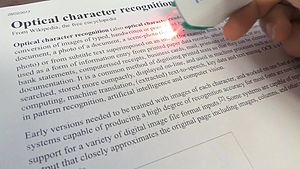
John Ball, cognitive scientist and inventor of Patom Theory supports this assessment. Natural language processing has made inroads for applications to support human productivity in service and ecommerce but this has largely been made possible by narrowing the scope of the application. There are thousands of ways to request something in a human language which still defies conventional natural language processing. "To have a meaningful conversation with machines is only possible when we match every word to the correct meaning based on the meanings of the other words in the sentence – just like a 3-year-old does without guesswork" Patom Theory.

The umbrella term "natural language understanding" can be applied to a diverse set of computer applications, ranging from small, relatively simple tasks such as short commands issued to robots, to highly complex endeavors such as the full comprehension of newspaper articles or poetry passages. Many real world applications fall between the two extremes, for instance text classification for the automatic analysis of emails and their routing to a suitable department in a corporation does not require in depth understanding of the text, but needs to deal with a much larger vocabulary and more diverse syntax than the management of simple queries to database tables with fixed schemata.

Throughout the years various attempts at processing natural language or *English-like* sentences presented to computers have taken place at varying degrees of complexity. Some attempts have not resulted in systems with deep understanding, but have helped overall system usability. For example, Wayne Ratliff originally developed the *Vulcan* program with an English-like syntax to mimic the English speaking computer in Star Trek. Vulcan later became the dBase system whose easy-to-use syntax effectively launched the personal computer database industry.Systems with an easy to use or *English like* syntax are, however, quite distinct from systems that use a rich lexicon and include an internal representation (often as first order logic) of the semantics of natural language sentences.

Hence the breadth and depth of "understanding" aimed at by a system determine both the complexity of the system (and the implied challenges) and the types of applications it can deal with. The "breadth" of a system is measured by the sizes of its vocabulary and grammar. The "depth" is measured by the degree to which its understanding approximates that of a fluent native speaker. At the narrowest and shallowest, *English-like* command interpreters require minimal complexity, but have a small range of applications. Narrow but deep systems explore and model mechanisms of understanding, but they still have limited application. Systems that attempt to understand the contents of a document such as a news release beyond simple keyword matching and to judge its suitability for a user are broader and require significant complexity, but they are still somewhat shallow. Systems that are both very broad and very deep are beyond the current state of the art.

1. **Optical Character Recognition**



**Optical character recognition** (also **optical character reader**, **OCR**) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast). It is widely used as a form of information entry from printed paper data records, whether passport documents, invoices, bank statements, computerised receipts, business cards, mail, printouts of static-data, or any suitable documentation. It is a common method of digitising printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs. Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual component.

1. **Textual Entailment**

**Textual entailment** (**TE**) in natural language processing is a directional relation between text fragments. The relation holds whenever the truth of one text fragment follows from another text. In the TE framework, the entailing and entailed texts are termed *text (t)* and *hypothesis (h)*, respectively. Textual entailment is not the same as pure logical entailment — it has a more relaxed definition: "*t* entails *h*" (*t* ⇒ *h*) if, typically, a human reading *t* would infer that *h* is most likely true. The relation is directional because even if "*t* entails *h*", the reverse "*h* entails *t*" is much less certain.[2][3

Determining whether this relationship holds is an informal task, one which sometimes overlaps with the formal tasks of formal semantics (satisfying a strict condition will usually imply satisfaction of a less strict conditioned); additionally, textual entailment partially subsumes word entailment.

Textual entailment can be illustrated with examples of three different relations:

An example of a **positive TE** (text entails hypothesis) is:

* text: *If you help the needy, God will reward you*.

hypothesis: *Giving money to a poor man has good consequences*.

An example of a **negative TE** (text contradicts hypothesis) is:

* text: *If you help the needy, God will reward you*.

hypothesis: *Giving money to a poor man has no consequences*.

An example of a **non-TE** (text does not entail nor contradict) is:

* text: *If you help the needy, God will reward you*.

hypothesis: *Giving money to a poor man will make you a better person*.

1. **Question Answering**

**Question answering** (**QA**) is a computer science discipline within the fields of information retrieval and natural language processing (NLP), which is concerned with building systems that automatically answer questions posed by humans in a natural language.

A QA implementation, usually a computer program, may construct its answers by querying a structured database of knowledge or information, usually a knowledge base. More commonly, QA systems can pull answers from an unstructured collection of natural language documents.

Some examples of natural language document collections used for QA systems include:

* a local collection of reference texts
* internal organization documents and web pages
* compiled newswire reports
* a set of Wikipedia pages
* a subset of World Wide Web pages

QA research attempts to deal with a wide range of question types including: fact, list, definition, *How*, *Why*, hypothetical, semantically constrained, and cross-lingual questions.

* *Closed-domain* question answering deals with questions under a specific domain (for example, medicine or automotive maintenance), and can be seen as an easier task because NLP systems can exploit domain-specific knowledge frequently formalized in ontologies. Alternatively, *closed-domain* might refer to a situation where only a limited type of questions are accepted, such as questions asking for descriptive rather than procedural information. QA systems in the context of machine reading applications have also been constructed in the medical domain, for instance related to Alzheimers disease
* *Open-domain* question answering deals with questions about nearly anything, and can only rely on general ontologies and world knowledge. On the other hand, these systems usually have much more data available from which to extract the answer.

1. **Speech Recognition**

**Speech recognition** is the inter-disciplinary sub-field of computational linguistics that develops methodologies and technologies that enables the recognition and translation of spoken language into text by computers. It is also known as "automatic speech recognition" (ASR), "computer speech recognition", or just "speech to text" (STT). It incorporates knowledge and research in the linguistics, computer science, and electrical engineering fields.

Some speech recognition systems require "training" (also called "enrollment") where an individual speaker reads text or isolated vocabulary into the system. The system analyzes the person's specific voice and uses it to fine-tune the recognition of that person's speech, resulting in increased accuracy. Systems that do not use training are called "speaker independent"[1] systems. Systems that use training are called "speaker dependent".

Speech recognition applications include voice user interfaces such as voice dialing (e.g. "Call home"), call routing (e.g. "I would like to make a collect call"), domotic appliance control, search (e.g. find a podcast where particular words were spoken), simple data entry (e.g., entering a credit card number), preparation of structured documents (e.g. a radiology report), speech-to-text processing (e.g., word processors or emails), and aircraft (usually termed direct voice input).

The term *voice recognition* or *speaker identification* refers to identifying the speaker, rather than what they are saying. Recognizing the speaker can simplify the task of translating speech in systems that have been trained on a specific person's voice or it can be used to authenticate or verify the identity of a speaker as part of a security process.

From the technology perspective, speech recognition has a long history with several waves of major innovations. Most recently, the field has benefited from advances in deep learning and big data. The advances are evidenced not only by the surge of academic papers published in the field, but more importantly by the worldwide industry adoption of a variety of deep learning methods in designing and deploying speech recognition systems. These speech industry players include Google, Microsoft, IBM, Baidu, Apple, Amazon, Nuance, SoundHound, iFLYTEK many of which have publicized the core technology in their speech recognition systems as being based on deep learning.

* 1. **Practical speech recognition**

The 1990s saw the first introduction of commercially successful speech recognition technologies. Two of the earliest products were Dragon Dictate, a consumer product released in 1990 and originally priced at $9,000, and a recognizer from Kurzweil Applied Intelligence released in 1987. AT&T deployed the Voice Recognition Call Processing service in 1992 to route telephone calls without the use of a human operator. The technology was developed by Lawrence Rabiner and others at Bell Labs. By this point, the vocabulary of the typical commercial speech recognition system was larger than the average human vocabulary. Raj Reddy's former student, Xuedong Huang, developed the Sphinx-II system at CMU. The Sphinx-II system was the first to do speaker-independent, large vocabulary, continuous speech recognition and it had the best performance in DARPA's 1992 evaluation. Handling continuous speech with a large vocabulary was a major milestone in the history of speech recognition. Huang went on to found the speech recognition group at Microsoft in 1993. Raj Reddy's student Kai-Fu Lee joined Apple where, in 1992, he helped develop a speech interface prototype for the Apple computer known as Casper.

Lernout & Hauspie, a Belgium-based speech recognition company, acquired several other companies, including Kurzweil Applied Intelligence in 1997 and Dragon Systems in 2000. The L&H speech technology was used in the Windows XP operating system. L&H was an industry leader until an accounting scandal brought an end to the company in 2001. The speech technology from L&H was bought by ScanSoft which became Nuance in 2005. Apple originally licensed software from Nuance to provide speech recognition capability to its digital assistant Siri.

In the 2000s DARPA sponsored two speech recognition programs: Effective Affordable Reusable Speech-to-Text (EARS) in 2002 and Global Autonomous Language Exploitation (GALE). Four teams participated in the EARS program: IBM, a team led by BBN with LIMSIand Univ. of Pittsburgh, Cambridge University, and a team composed of ISCI, SRI and University of Washington. EARS funded the collection of the Switchboard telephone speech corpus containing 260 hours of recorded conversations from over 500 speakers. The GALE program focused on Arabic and Mandarin broadcast news speech. Google's first effort at speech recognition came in 2007 after hiring some researchers from Nuance. The first product was GOOG-411, a telephone based directory service. The recordings from GOOG-411 produced valuable data that helped Google improve their recognition systems. Google voice search is now supported in over 30 languages.

In the United States, the National Security Agency has made use of a type of speech recognition for keyword spotting since at least 2006.[36] This technology allows analysts to search through large volumes of recorded conversations and isolate mentions of keywords. Recordings can be indexed and analysts can run queries over the database to find conversations of interest. Some government research programs focused on intelligence applications of speech recognition, e.g. DARPA's EARS's program and IARPA's Babel program.

* 1. **Modern systems**

In the early 2000s, speech recognition was still dominated by traditional approaches such as Hidden Markov Models combined with feedforward artificial neural networks.[37] Today, however, many aspects of speech recognition have been taken over by a deep learningmethod called Long short-term memory (LSTM), a recurrent neural network published by Sepp Hochreiter & Jürgen Schmidhuber in 1997. LSTM RNNs avoid the vanishing gradient problem and can learn "Very Deep Learning" tasks that require memories of events that happened thousands of discrete time steps ago, which is important for speech. Around 2007, LSTM trained by Connectionist Temporal Classification (CTC) started to outperform traditional speech recognition in certain applications. In 2015, Google's speech recognition reportedly experienced a dramatic performance jump of 49% through CTC-trained LSTM, which is now available through Google Voice to all smartphone users.

The use of deep feedforward (non-recurrent) networks for acoustic modeling was introduced during later part of 2009 by Geoffrey Hinton and his students at University of Toronto and by Li Deng and colleagues at Microsoft Research, initially in the collaborative work between Microsoft and University of Toronto which was subsequently expanded to include IBM and Google (hence "The shared views of four research groups" subtitle in their 2012 review paper). A Microsoft research executive called this innovation "the most dramatic change in accuracy since 1979.".In contrast to the steady incremental improvements of the past few decades, the application of deep learning decreased word error rate by 30%. This innovation was quickly adopted across the field. Researchers have begun to use deep learning techniques for language modeling as well.

In the long history of speech recognition, both shallow form and deep form (e.g. recurrent nets) of artificial neural networks had been explored for many years during 1980s, 1990s and a few years into the 2000s. But these methods never won over the non-uniform internal-handcrafting Gaussian mixture model/Hidden Markov model (GMM-HMM) technology based on generative models of speech trained discriminatively. A number of key difficulties had been methodologically analyzed in the 1990s, including gradient diminishing and weak temporal correlation structure in the neural predictive models. All these difficulties were in addition to the lack of big training data and big computing power in these early days. Most speech recognition researchers who understood such barriers hence subsequently moved away from neural nets to pursue generative modeling approaches until the recent resurgence of deep learning starting around 2009–2010 that had overcome all these difficulties. Hinton et al. and Deng et al. reviewed part of this recent history about how their collaboration with each other and then with colleagues across four groups (University of Toronto, Microsoft, Google, and IBM) ignited a renaissance of applications of deep feedforward neural networks to speech recognition.

1. **Speech Segmentation**

**Speech segmentation** is the process of identifying the boundaries between words, syllables, or phonemes in spoken natural languages. The term applies both to the mental processes used by humans, and to artificial processes of natural language processing.

Speech segmentation is a subfield of general speech perception and an important subproblem of the technologically focused field of speech recognition, and cannot be adequately solved in isolation. As in most natural language processing problems, one must take into account context, grammar, and semantics, and even so the result is often a probabilistic division (statistically based on likelihood) rather than a categorical one. Though it seems that coarticulation—a phenomenon which may happen between adjacent words just as easily as within a single word—presents the main challenge in speech segmentation across languages, some other problems and strategies employed in solving those problems can be seen in the following sections.

This problem overlaps to some extent with the problem of text segmentation that occurs in some languages which are traditionally written without inter-word spaces, like Chinese and Japanese, compared to writing systems which indicate speech segmentation between words by a word divider, such as the space. However, even for those languages, text segmentation is often much easier than speech segmentation, because the written language usually has little interference between adjacent words, and often contains additional clues not present in speech (such as the use of Chinese characters for word stems in Japanese). Word Boundary Identification can be overcome by NLU approaches such as Patom theory integrated with Role and Reference Grammar (RRG) for languages without spaces between words such as Japanese and Chinese.

In natural languages, the meaning of a complex spoken sentence can be understood by decomposing it into smaller lexical segments (roughly, the words of the language), associating a meaning to each segment, and combining those meanings according to the grammar rules of the language.

Though lexical recognition is not thought to be used by infants in their first year, due to their highly limited vocabularies, it is one of the major processes involved in speech segmentation for adults. Three main models of lexical recognition exist in current research: first, whole-word access, which argues that words have a whole-word representation in the lexicon; second, decomposition, which argues that morphologically complex words are broken down into their morphemes (roots, stems, inflections, etc.) and then interpreted and; third, the view that whole-word and decomposition models are both used, but that the whole-word model provides some computational advantages and is therefore dominant in lexical recognition.

To give an example, in a whole-word model, the word "cats" might be stored and searched for by letter, first "c", then "ca", "cat", and finally "cats". The same word, in a decompositional model, would likely be stored under the root word "cat" and could be searched for after removing the "s" suffix. "Falling", similarly, would be stored as "fall" and suffixed with the "ing" inflection.

Though proponents of the decompositional model recognize that a morpheme-by-morpheme analysis may require significantly more computation, they argue that the unpacking of morphological information is necessary for other processes (such as syntactic structure) which may occur parallel to lexical searches.

As a whole, research into systems of human lexical recognition is limited due to little experimental evidence that fully discriminates between the three main models.[1]

In any case, lexical recognition likely contributes significantly to speech segmentation through the contextual clues it provides, given that it is a heavily probabilistic system—based on the statistical likelihood of certain words or constituents occurring together. For example, one can imagine a situation where a person might say "I bought my dog at a \_\_\_\_ shop" and the missing word's vowel is pronounced as in "net", "sweat", or "pet". While the probability of "netshop" is extremely low, since "netshop" isn't currently a compound or phrase in English, and "sweatshop" also seems contextually improbable, "pet shop" is a good fit because it is a common phrase and is also related to the word "dog".

Moreover, an utterance can have different meanings depending on how it is split into words. A popular example, often quoted in the field, is the phrase "How to wreck a nice beach", which sounds very similar to "How to recognize speech". As this example shows, proper lexical segmentation depends on context and semantics which draws on the whole of human knowledge and experience, and would thus require advanced pattern recognition and artificial intelligence technologies to be implemented on a computer.

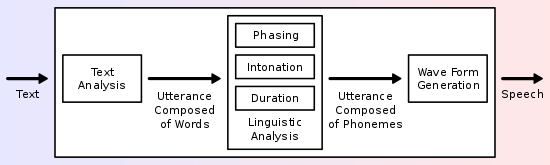
Lexical recognition is of particular value in the field of computer speech recognition, since the ability to build and search a network of semantically connected ideas would greatly increase the effectiveness of speech-recognition software. Statistical models can be used to segment and align recorded speech to words or phones. Applications include automatic lip-synch timing for cartoon animation, follow-the-bouncing-ball video sub-titling, and linguistic research. Automatic segmentation and alignment software is commercially available.

1. **Speech Synthesis**

**Speech synthesis** is the artificial production of human speech. A computer system used for this purpose is called a **speech computer** or **speech synthesizer**, and can be implemented in software or hardware products. A **text-to-speech** (**TTS**) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech.

Synthesized speech can be created by concatenating pieces of recorded speech that are stored in a database. Systems differ in the size of the stored speech units; a system that stores phones or diphones provides the largest output range, but may lack clarity. For specific usage domains, the storage of entire words or sentences allows for high-quality output. Alternatively, a synthesizer can incorporate a model of the vocal tract and other human voice characteristics to create a completely "synthetic" voice output.

The quality of a speech synthesizer is judged by its similarity to the human voice and by its ability to be understood clearly. An intelligible text-to-speech program allows people with visual impairments or reading disabilities to listen to written words on a home computer. Many computer operating systems have included speech synthesizers since the early 1990s.

[](https://en.wikipedia.org/wiki/File:TTS_System.svg)

A text-to-speech system (or "engine") is composed of two parts: a front-end and a back-end. The front-end has two major tasks. First, it converts raw text containing symbols like numbers and abbreviations into the equivalent of written-out words. This process is often called *text normalization*, *pre-processing*, or *tokenization*. The front-end then assigns phonetic transcriptions to each word, and divides and marks the text into prosodic units, like phrases, clauses, and sentences. The process of assigning phonetic transcriptions to words is called *text-to-phoneme* or *grapheme-to-phoneme* conversion. Phonetic transcriptions and prosody information together make up the symbolic linguistic representation that is output by the front-end. The back-end—often referred to as the *synthesizer*—then converts the symbolic linguistic representation into sound. In certain systems, this part includes the computation of the *target prosody* (pitch contour, phoneme durations) which is then imposed on the output speech.

The most important qualities of a speech synthesis system are *naturalness* and *intelligibility*. Naturalness describes how closely the output sounds like human speech, while intelligibility is the ease with which the output is understood. The ideal speech synthesizer is both natural and intelligible. Speech synthesis systems usually try to maximize both characteristics.

The two primary technologies generating synthetic speech waveforms are *concatenative synthesis* and *formant synthesis*. Each technology has strengths and weaknesses, and the intended uses of a synthesis system will typically determine which approach is used.

**Concatenation synthesis**

Concatenative synthesis is based on the concatenation (or stringing together) of segments of recorded speech. Generally, concatenative synthesis produces the most natural-sounding synthesized speech. However, differences between natural variations in speech and the nature of the automated techniques for segmenting the waveforms sometimes result in audible glitches in the output. There are three main sub-types of concatenative synthesis.

**Unit selection synthesis**

Unit selection synthesis uses large databases of recorded speech. During database creation, each recorded utterance is segmented into some or all of the following: individual phones, diphones, half-phones, syllables, morphemes, words, phrases, and sentences. Typically, the division into segments is done using a specially modified speech recognizer set to a "forced alignment" mode with some manual correction afterward, using visual representations such as the waveform and spectrogram. An index of the units in the speech database is then created based on the segmentation and acoustic parameters like the fundamental frequency (pitch), duration, position in the syllable, and neighboring phones. At run time, the desired target utterance is created by determining the best chain of candidate units from the database (unit selection). This process is typically achieved using a specially weighted decision tree.

Unit selection provides the greatest naturalness, because it applies only a small amount of digital signal processing (DSP) to the recorded speech. DSP often makes recorded speech sound less natural, although some systems use a small amount of signal processing at the point of concatenation to smooth the waveform. The output from the best unit-selection systems is often indistinguishable from real human voices, especially in contexts for which the TTS system has been tuned. However, maximum naturalness typically require unit-selection speech databases to be very large, in some systems ranging into the gigabytes of recorded data, representing dozens of hours of speech. Also, unit selection algorithms have been known to select segments from a place that results in less than ideal synthesis (e.g. minor words become unclear) even when a better choice exists in the database. Recently, researchers have proposed various automated methods to detect unnatural segments in unit-selection speech synthesis systems.

**Diphone synthesis**

Diphone synthesis uses a minimal speech database containing all the diphones (sound-to-sound transitions) occurring in a language. The number of diphones depends on the phonotactics of the language: for example, Spanish has about 800 diphones, and German about 2500. In diphone synthesis, only one example of each diphone is contained in the speech database. At runtime, the target prosody of a sentence is superimposed on these minimal units by means of digital signal processing techniques such as linear predictive coding, PSOLA or MBROLA.or more recent techniques such as pitch modification in the source domain using discrete cosine transform Diphone synthesis suffers from the sonic glitches of concatenative synthesis and the robotic-sounding nature of formant synthesis, and has few of the advantages of either approach other than small size. As such, its use in commercial applications is declining, although it continues to be used in research because there are a number of freely available software implementations.

**Domain-specific synthesis**

Domain-specific synthesis concatenates prerecorded words and phrases to create complete utterances. It is used in applications where the variety of texts the system will output is limited to a particular domain, like transit schedule announcements or weather reports.The technology is very simple to implement, and has been in commercial use for a long time, in devices like talking clocks and calculators. The level of naturalness of these systems can be very high because the variety of sentence types is limited, and they closely match the prosody and intonation of the original recordings.

Because these systems are limited by the words and phrases in their databases, they are not general-purpose and can only synthesize the combinations of words and phrases with which they have been preprogrammed. The blending of words within naturally spoken language however can still cause problems unless the many variations are taken into account. For example, in non-rhotic dialects of English the *"r"* in words like *"clear"* /ˈklɪə/ is usually only pronounced when the following word has a vowel as its first letter (e.g. *"clear out"*is realized as /ˌklɪəɾˈʌʊt/). Likewise in French, many final consonants become no longer silent if followed by a word that begins with a vowel, an effect called liaison. This alternation cannot be reproduced by a simple word-concatenation system, which would require additional complexity to be context-sensitive.

**Formant synthesis**

Formant synthesis does not use human speech samples at runtime. Instead, the synthesized speech output is created using additive synthesis and an acoustic model (physical modelling synthesis). Parameters such as fundamental frequency, voicing, and noise levels are varied over time to create a waveform of artificial speech. This method is sometimes called *rules-based synthesis*; however, many concatenative systems also have rules-based components. Many systems based on formant synthesis technology generate artificial, robotic-sounding speech that would never be mistaken for human speech. However, maximum naturalness is not always the goal of a speech synthesis system, and formant synthesis systems have advantages over concatenative systems. Formant-synthesized speech can be reliably intelligible, even at very high speeds, avoiding the acoustic glitches that commonly plague concatenative systems. High-speed synthesized speech is used by the visually impaired to quickly navigate computers using a screen reader. Formant synthesizers are usually smaller programs than concatenative systems because they do not have a database of speech samples. They can therefore be used in embedded systems, where memory and microprocessor power are especially limited. Because formant-based systems have complete control of all aspects of the output speech, a wide variety of prosodies and intonations can be output, conveying not just questions and statements, but a variety of emotions and tones of voice.

Examples of non-real-time but highly accurate intonation control in formant synthesis include the work done in the late 1970s for the Texas Instruments toy Speak & Spell, and in the early 1980s Sega arcade machines and in many Atari, Inc. arcade games[32] using the TMS5220 LPC Chips. Creating proper intonation for these projects was painstaking, and the results have yet to be matched by real-time text-to-speech interfaces

Formant synthesis was implemented in hardware in the Yamaha FS1R synthesizer, but the speech aspect of formants was never realized in the synth. It was capable of short, several-second formant sequences which could speak a single phrase, but since the MIDI control interface was so restrictive live speech was an impossibility.

**IV.**

**EXPERIMENTATION**

**&**

**RESULT**

**4.1 PROCEDURE**

INTRODUCTION

ENTER DETAILS

IF PATIENT DETAILS RYT

OTHERWISE

SYSTEM

FAILED

SCHEMES WOULD BE READ OUT AND SHOWN

Firstly we enter user details i.e. name and age then the application ask the user if he wants to read the schemes in textual form or he wants to listen the schemes in audio format. On users choice the application performs the desired action.

For the textual format the pdf is first converted to txt format and then shown to the user. For audio format the Text To Speech function is used.

**SOFTWARE SPECIFICATION**

1.JAVA- **Java** is a general-purpose [computer-programming language](https://en.wikipedia.org/wiki/Programming_language) that is [concurrent](https://en.wikipedia.org/wiki/Concurrent_computing), [class-based](https://en.wikipedia.org/wiki/Class-based_programming), [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "[write once, run anywhere](https://en.wikipedia.org/wiki/Write_once,_run_anywhere)" (WORA), meaning that [compiled](https://en.wikipedia.org/wiki/Compiler) Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to [bytecode](https://en.wikipedia.org/wiki/Java_bytecode) that can run on any [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) (JVM) regardless of [computer architecture](https://en.wikipedia.org/wiki/Computer_architecture). As of 2016, Java is one of the most [popular programming languages in use](https://en.wikipedia.org/wiki/Measuring_programming_language_popularity), particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by [James Gosling](https://en.wikipedia.org/wiki/James_Gosling) at [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems) (which has since been [acquired by Oracle Corporation](https://en.wikipedia.org/wiki/Sun_acquisition_by_Oracle)) and released in 1995 as a core component of Sun Microsystems' [Java platform](https://en.wikipedia.org/wiki/Java_(software_platform)). The language derives much of its [syntax](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) from [C](https://en.wikipedia.org/wiki/C_(programming_language)) and [C++](https://en.wikipedia.org/wiki/C%2B%2B), but it has fewer [low-level](https://en.wikipedia.org/wiki/Low-level_programming_language) facilities than either of them.

The original and [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) Java [compilers](https://en.wikipedia.org/wiki/Compiler), virtual machines, and [class libraries](https://en.wikipedia.org/wiki/Library_(computing)) were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the [Java Community Process](https://en.wikipedia.org/wiki/Java_Community_Process), Sun [relicensed](https://en.wikipedia.org/wiki/Software_relicensing) most of its Java technologies under the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License). Others have also developed alternative implementations of these Sun technologies, such as the [GNU Compiler for Java](https://en.wikipedia.org/wiki/GNU_Compiler_for_Java) (bytecode compiler), [GNU Classpath](https://en.wikipedia.org/wiki/GNU_Classpath) (standard libraries), and [IcedTea](https://en.wikipedia.org/wiki/IcedTea)-Web (browser plugin for applets).

The latest version is [Java 10](https://en.wikipedia.org/wiki/Java_version_history), released on March 20, 2018, which follows [Java 9](https://en.wikipedia.org/wiki/Java_version_history) after only six months in line with the new release schedule. Java 8 is still supported but there will be no more security updates for Java 9. Versions earlier than Java 8 are supported by companies on a commercial basis; e.g. by Oracle back to Java 6 as of October 2017 (while they still "highly recommend that you uninstall" pre-Java 8 from at least Windows computers).

2. J2ME- **Java Platform, Micro Edition** or **Java ME** is a [computing platform](https://en.wikipedia.org/wiki/Computing_platform) for development and deployment of [portable code](https://en.wikipedia.org/wiki/Porting) for [embedded](https://en.wikipedia.org/wiki/Embedded_system) and [mobile devices](https://en.wikipedia.org/wiki/Mobile_device) (micro-controllers, sensors, gateways, mobile phones, personal digital assistants, TV set-top boxes, printers). Java ME was formerly known as **Java 2 Platform, Micro Edition** or **J2ME**.

The platform uses the [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) programming language. It is part of the [Java software-platform](https://en.wikipedia.org/wiki/Java_(software_platform)) family. Java ME was designed by [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems), acquired by [Oracle Corporation](https://en.wikipedia.org/wiki/Oracle_Corporation) in 2010; the platform replaced a similar technology, [PersonalJava](https://en.wikipedia.org/wiki/PersonalJava). Originally developed under the [Java Community Process](https://en.wikipedia.org/wiki/Java_Community_Process) as JSR 68, the different flavors of Java ME have evolved in separate JSRs. Oracle provides a [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of the specification, but has tended not to provide free binary implementations of its Java ME runtime environment for mobile devices, rather relying on third parties to provide their own.

As of 22 December 2006, the Java ME source code is licensed under the [GNU General Public License](https://en.wikipedia.org/wiki/GNU_General_Public_License), and is released under the project name [phone ME](https://en.wikipedia.org/wiki/PhoneME).

As of 2008, all Java ME platforms are currently restricted to [JRE](https://en.wikipedia.org/wiki/Java_Virtual_Machine) 1.3 features and use that version of the class file format (internally known as version 47.0). Should Oracle ever declare a new round of Java ME configuration versions that support the later class file formats and language features, such as those corresponding to JRE 1.5 or 1.6 (notably, [generics](https://en.wikipedia.org/wiki/Generics_in_Java)), it will entail extra work on the part of all platform vendors to update their JREs.

Java ME devices implement a *profile*. The most common of these are the [Mobile Information Device Profile](https://en.wikipedia.org/wiki/Mobile_Information_Device_Profile) aimed at mobile devices, such as cell phones, and the [Personal Profile](https://en.wikipedia.org/wiki/Personal_Profile) aimed at consumer products and embedded devices like [set-top boxes](https://en.wikipedia.org/wiki/Set-top_box) and PDAs. Profiles are subsets of *configurations*, of which there are currently two: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC).[[2]](https://en.wikipedia.org/wiki/Java_Platform,_Micro_Edition#cite_note-2)

There are more than 2.1 billion Java ME enabled mobile phones and PDAs. It is popular in sub $200 devices such as Nokia's [Series 40](https://en.wikipedia.org/wiki/Series_40). It was also used on the [Bada](https://en.wikipedia.org/wiki/Bada) operating system and on [Symbian](https://en.wikipedia.org/wiki/Symbian) OS along with native software. Users of [Windows CE](https://en.wikipedia.org/wiki/Windows_CE), [Windows Mobile](https://en.wikipedia.org/wiki/Windows_Mobile), [Maemo](https://en.wikipedia.org/wiki/Maemo), [MeeGo](https://en.wikipedia.org/wiki/MeeGo) and [Android](https://en.wikipedia.org/wiki/Android_software_development) can download Java ME for their respective environments.

3.**Android** **SDK**- The Android [software development kit](https://en.wikipedia.org/wiki/Software_development_kit) (SDK) includes a comprehensive set of development tools.] These include a [debugger](https://en.wikipedia.org/wiki/Debugger), [libraries](https://en.wikipedia.org/wiki/Software_library), a handset [emulator](https://en.wikipedia.org/wiki/Emulator) based on [QEMU](https://en.wikipedia.org/wiki/QEMU), documentation, sample code, and tutorials. Currently supported development platforms include computers running [Linux](https://en.wikipedia.org/wiki/Linux_kernel) (any modern desktop [Linux distribution](https://en.wikipedia.org/wiki/List_of_Linux_distributions)), [Mac OS X](https://en.wikipedia.org/wiki/Mac_OS_X) 10.5.8 or later, and [Windows 7](https://en.wikipedia.org/wiki/Windows_7) or later. As of March 2015, the SDK is not available on Android itself, but software development is possible by using specialized Android applications.

Until around the end of 2014, the officially supported [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) was [Eclipse](https://en.wikipedia.org/wiki/Eclipse_(software)) using the Android Development Tools (ADT) Plugin, though [IntelliJ IDEA](https://en.wikipedia.org/wiki/IntelliJ_IDEA) IDE (all editions) fully supports Android development out of the box,and [NetBeans](https://en.wikipedia.org/wiki/NetBeans) IDE also supports Android development via a pluginAs of 2015, [Android Studio](https://en.wikipedia.org/wiki/Android_Studio) made by Google and powered by IntelliJ, is the official IDE; however, developers are free to use others, but Google made it clear that ADT was officially deprecated since the end of 2015 to focus on Android Studio as the official Android IDE.Additionally, developers may use any text editor to edit Java and XML files, then use [command line](https://en.wikipedia.org/wiki/Command_line) tools ([Java Development Kit](https://en.wikipedia.org/wiki/Java_Development_Kit) and [Apache Ant](https://en.wikipedia.org/wiki/Apache_Ant) are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely).]

Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

Android applications are packaged in [.apk](https://en.wikipedia.org/wiki/APK_(file_format)) format and stored under /data/app folder on the Android OS (the folder is accessible only to the root user for security reasons). APK package contains .dex files (compiled byte code files called [Dalvik](https://en.wikipedia.org/wiki/Dalvik_Virtual_Machine) executables), resource files, etc.

**Android Debug Bridge**[[edit](https://en.wikipedia.org/w/index.php?title=Android_software_development&action=edit&section=3)]

The Android Debug Bridge (ADB) is a toolkit included in the Android SDK package. It consists of both client and server-side programs that communicate with one another. The ADB is typically accessed through the [command-line interface](https://en.wikipedia.org/wiki/Command-line_interface), although numerous [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface) exist to control ADB.

The format for issuing commands through the ADB is typically:

adb [-d|-e|-s <serialNumber>] <command>

where -d is the option for specifying the USB-attached device,

-e for indicating a running Android emulator on the computer,

-s for specifying either one by its adb-assigned serial number.

If there is only one attached device or running emulator, these options are not necessary.

For example, Android [applications](https://en.wikipedia.org/wiki/Android_application_package) can be saved by the command [backup](https://en.wikipedia.org/wiki/Backup) to a file, whose name is backup.ab by default.

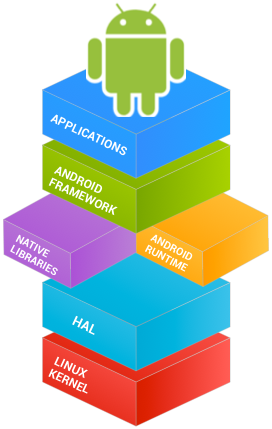
In a security issue reported in March 2011, ADB was targeted as a vector to attempt to install a rootkit on connected phones using a "resource exhaustion attack".

**Fastboot**

*"Fastboot" redirects here. For the PC fast booting ability, see*[*Instant-on*](https://en.wikipedia.org/wiki/Instant-on)*.*

*Fastboot* is a diagnostic [protocol](https://en.wikipedia.org/wiki/Communications_protocol) included with the SDK package used primarily to modify the [flash](https://en.wikipedia.org/wiki/Flash_memory) [filesystem](https://en.wikipedia.org/wiki/Filesystem) via a [USB](https://en.wikipedia.org/wiki/USB) connection from host computer. It requires that the device be started in a [boot loader](https://en.wikipedia.org/wiki/Boot_loader) or [Secondary Program Loader](https://en.wikipedia.org/wiki/Secondary_Program_Loader) mode, in which only the most basic hardware initialization is performed. After enabling the protocol on the device itself, it will accept a specific set of commands sent to it via USB using a command line. Some of the most commonly used fastboot commands include:

* flash – rewrites a partition with a binary image stored on the host computer
* erase – erases a specific partition
* reboot – reboots the device into either the main operating system, the system recovery partition or back into its boot loader
* devices – displays a list of all devices (with the serial number) connected to the host computer
* format – formats a specific partition; the file system of the partition must be recognized by the device

**Android Stack**

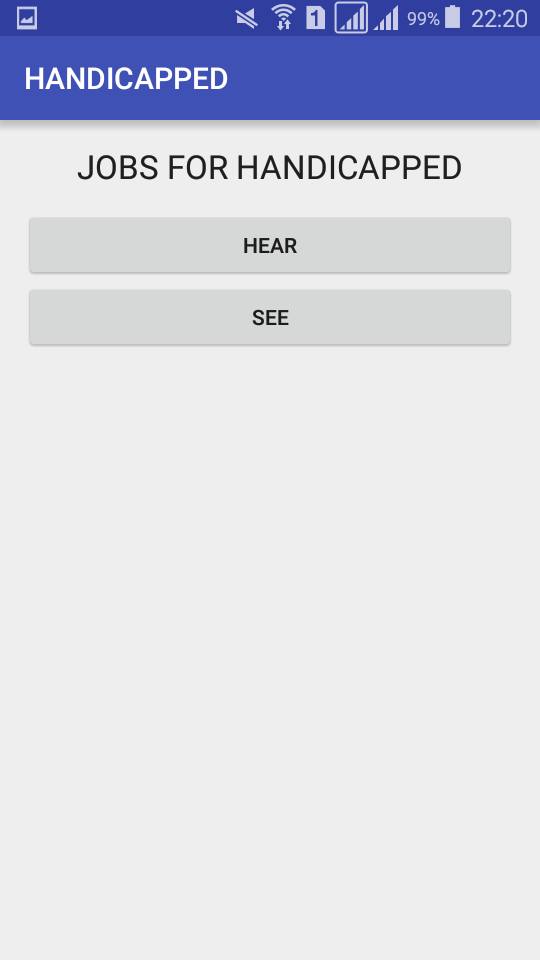


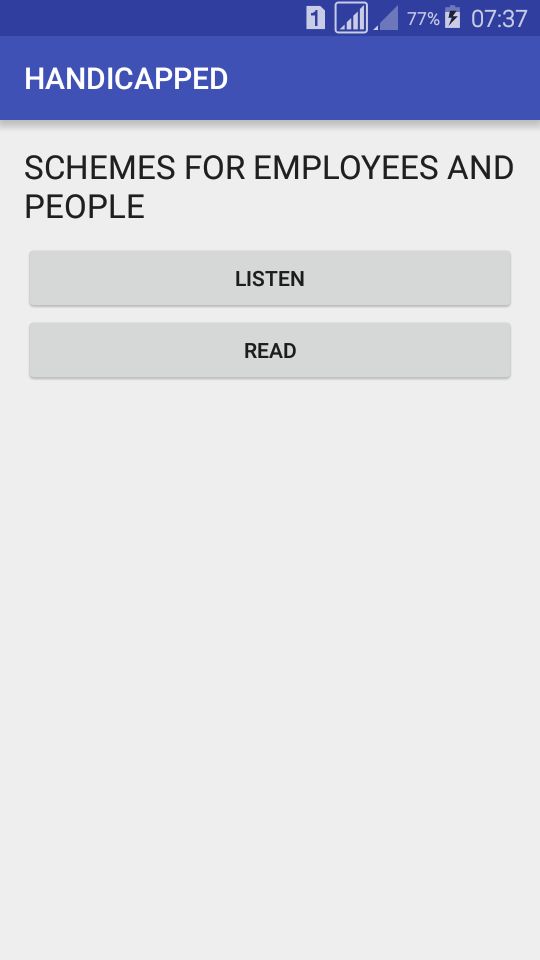
**Java**

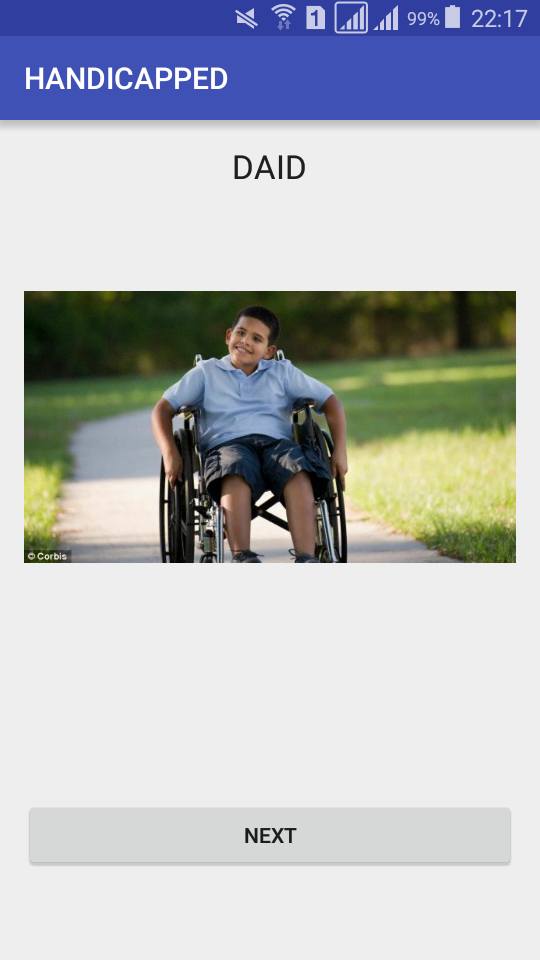
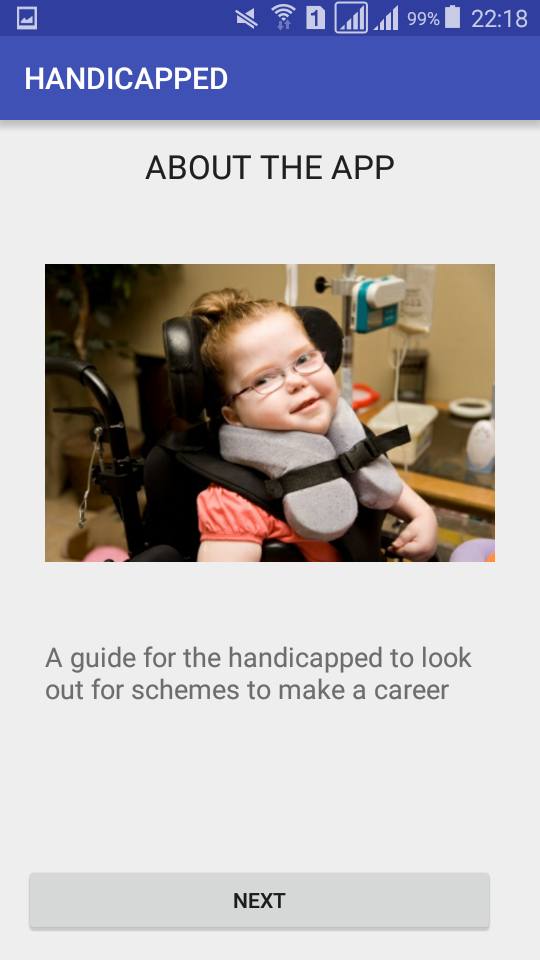


**4.2 RESULTS**

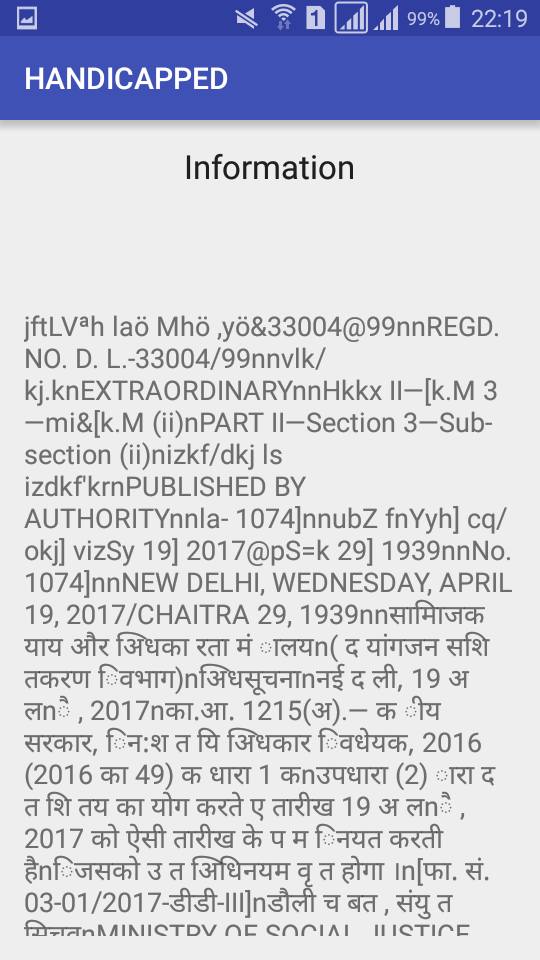
We are able to achieve following results









**V.FURTHER**

**V.**

**IMPROVEMENTS**

**AND**

**CONCLUSIONS**

**5.1 SCOPE OF IMPROVEMENT**

We can integrate a chatbot in our android application in which user can enquire about the schemes that are being told to him . We can also integrate a language input type in which user can enter the language of his choice and audio will be played in that language . Similar thing can take place in case of a chatbot in which user can select language of his choice and type his query in that language. Hence his query will also be replied in that language.

**5.2 CONCLUSION**

Hence we are successful in developing an android application which would explore the scope of physically handicapped under gazette notification.

**References**

**References**

* + - 1. 1. Development of Input Assistance Application for Mobile Devices for Physically Disabled
      2. by Yuki Sarugi Deptt of ECE.
      3. 2. Chatbot Using A Knowledge in Database Human-to-Machine Conversation Modeling by
      4. Ferry Wahyu Wibowo Department of Informatics Engineering.
      5. 3. Towards an efficient voice-based chatbot by J. Quintero, Student Member, IEEE, and R.
      6. Asprilla, Member, IEEE.
      7. 4. Android Based Speech Recognition by Sonali Thite
      8. 5. A mobile communication aid for physically handicapped by Cheng Huei Yanga.
      9. 6. Voice based online examination for physically handicapped by Sania Khan Dept. of CS & E Moradabad Institute of Technology Moradabad, U.P., India
      10. 7. Design and Development of Voice Activated Intelligent System for Elderly and Physically
      11. Challenged by Raju Hajare Dept. of TCE, BMSIT, Bangalore .
      12. 8. Study and Development of Support Tool with Blinks for Physically Handicapped
      13. Children by Ippei Torii Department of Information Science Aichi Institute of Technology
      14. Aichi, Japan and Takahito Niwa Department of Information Science Aichi Institute of
      15. Technology Aichi, Japan.