Approaches towards Building a Banking Assistant

Kiner B. Shah
Department of Computer Engineering
K. J. Somaiya College of Engineering, Mumbai, India

Mohit S. Shetty
Department of Computer Engineering
K. J. Somaiya College of Engineering, Mumbai, India

Darshan P. Shah
Department of Computer Engineering
K. J. Somaiya College of Engineering, Mumbai, India

Rajni Pamnani
Assistant Professor, Department of Computer Engineering
K. J. Somaiya College of Engineering, Mumbai, India

ABSTRACT
Banking process has been very complicated since years. People often want to enquire about bank's policies on the bank counter and since the policies are sometimes confusing, it takes time for them to understand the policy and thus, the process. This paper presents two approaches - one using Natural Language Processing techniques and other using AIML, a popular language for building chatbots - for building banking assistant which can solve people's queries and also carry out certain banking tasks, thus avoiding loss of efficiency and loss of precious time of the people. The paper is aimed at providing interface to the users which enables communication for solving their queries and completing their tasks, thus saving their time and reducing any possible confusion.

Keywords
Natural Language Processing, Assistant, Artificial Intelligence Markup Language (AIML), Chatbot

1. INTRODUCTION
Banking is a process which is done daily by hundreds of people, some making certain transactions like depositing funds or withdrawing funds, some opening accounts whether it is a current account or a fixed deposit account, some applying for loans for car, home, or business, some submitting applications for issue of new cheque books, and some just go to the bank for enquiring about certain policies, certain issues, or for issuing complaints.

There are lots of such tasks that are time consuming and complicated to understand. Even if the policies are well documented, understanding the documentation in itself is a very tedious task. One person encounters with jargon terms, complicated clauses, certain confusing disclaimers, etc. Understanding these for a common person is as tedious as pulling a truck full of black soil. Other problems include inability to understand process by unlettered people and also language barrier which is common in a diverse nation like India.

This paper thus proposes architectures based on recent research in AI and NLP techniques, which will serve as an assistant for common people and which will efficiently help them to carry out their banking related tasks. *Natural Language Processing (NLP)* involves techniques to intelligently process natural language input making it easier for the computer for further processing.

The concept of chatbot using AIML / XML came into existence with "Alice" [18] which is used to receive question from user and it was based on pattern recognition. With the advent of time, there were many developments in this field and many more bots were built using the AIML methodology. In this approach AIML knowledge base is the base of Chatbot brain.

The next section presents some already developed assistants. Then some potential problems during the designing phase are discussed. Then finally the architectures of the assistant are presented which includes the two approaches of building assistant.

2. RELATED WORK
There are many projects implemented earlier for banking assistant. First system is Royal Bank of Scotland's "*Luvo*" [1]. Luvo, is able to understand questions and then filter through huge amounts of information in a split second before responding with the answer. If Luvo is unable to find the answer, it passes the query on to a member of staff who can solve more complex problems. It will support staff to help them answer customer queries more quickly and easily. It has to be trained when dealing with new subject matter, but crucially, it learns from its mistakes and its answers become more accurate over time. One problem with Luvo, however, is that it can interact only with bank staff and not directly with customers. Also, there is still the need for human experts despite having AI deployed.

Kasisto's AI (KAI) banking [2] is a conversational AI platform which makes engaging with customers as natural as chatting. The AI has deep financial knowledge and banking is easy from mundane tasks to complicated tasks – it's basically sending text messages. It is a very fine tool and does most of the tasks which is required for general banking like checking for previous transactions, making payments, telling information about accounts and the main point is that it is always learning. It chats with its users in a friendly manner. It is a good implementation of banking assistant.

Another such assistant was suggested [3] which was implemented as a web service (based on black-box approach) able to process multiple client requests simultaneously and which generated responses by using a data repository (based on AIML).

3. DESIGN ISSUES
Designing assistant software which can communicate as naturally as a human assistant does is not an easy task! There are lots of issues which may come while designing the assistant for a bank:

- There may be huge computational overheads because of which large memories, fast processors and sometimes, parallel architectures are required.

- Also, there is an issue in choosing an appropriate interface style – voice-based or text-based or both.
- It is difficult to design assistant for a particular domain like banking, as the data required for it can be costly or have to be made manually which is time consuming.
- It consumes lots of time to build an assistant, and if the other competitors launch a similar product before the bank does, then it creates a business risk for the bank.
- Also, there has to be sufficient investment in designing the assistant. Engineers have to be hired, tools need to be purchased, licensing, etc. and it may get very costly for the bank.
- If the assistant is dealing with transactions, then there is always an issue of providing appropriate security features.
- There is also a problem of choosing how to implement the assistant. There are two ways: rule-based and NLP-based. We will shortly present the differences between these two.

Some of the (but not all) design issues in designing assistant software for a bank have been discussed. The architecture which will be presented in the next section will try to consider some of the above issues.

4. ARCHITECTURE

In this section, firstly, some of the key differences in the two approaches of implementing the assistant – the *Rule-based approach* and the *NLP-based approach,* will be discussed. Then the architecture for both the approaches will be presented.

4.1 Rule-based Vs. NLP-based

In rule-based approach, there are various rules or patterns defined for the users' queries. Based, on what rule or pattern is matched, an appropriate predefined and stored answer is output or some pattern is defined for the answer which may use some information from the query itself. One very good example of rule-based or pattern-based approach is *ALICE* [4], which uses *AIML* [5] as the language for defining patterns for queries and its answers. The main limitation of this approach, however, is that large numbers of rules are required for ensuring proper working of the assistant. However, this approach is widely used in building various assistants. Some examples of assistants using this approach are, ALICE, Chatterbot, Jabberwacky, etc. If combined with machine learning, this approach can significantly improve the efficiency of the assistant.

In NLP-based approach, various NLP techniques are applied to process the sentence, check its grammar, extract information out of it, and use this information extracted for analysis and then generate an appropriate response with respect to the query. Hence, this approach becomes quite complicated and complex. A very good example of NLP-based assistant is IBM's Watson [6]. It uses many NLP and Machine learning techniques like logistic regression, Named Entity Recognition, Co-reference resolution, Relation Extraction, etc. It has a very sophisticated architecture which extracts data from large number of data sources and uses very high performance hardware which includes racks of servers, TBs of RAM, thousands of processor cores, and is capable of operating at about 80 teraflops. This led to it winning the Jeopardy challenge in 2011. One limitation of this approach is that this approach is slower if the architectures are not parallelized.

4.2 Architecture using NLP-based approach

Now, that the two approaches have been discussed, the architecture proposed for banking assistant will be presented which follows a NLP-based approach. The entire architecture is shown in Fig 1.

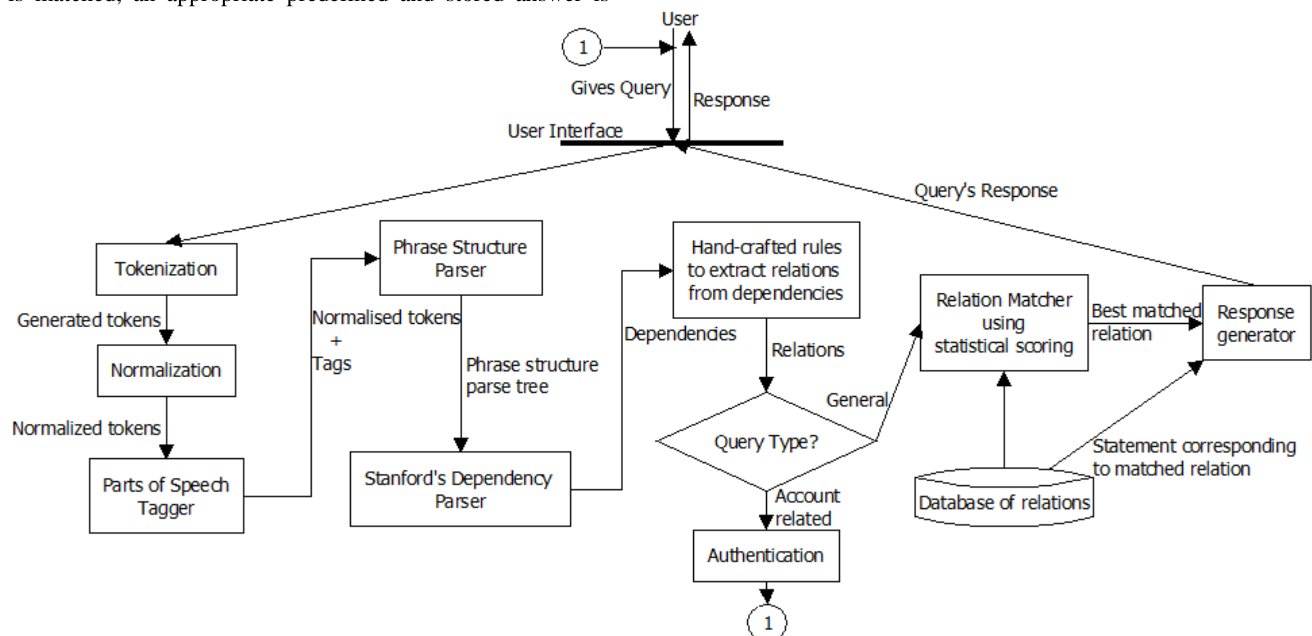

Fig 1: Architecture of NLP-based approach

In the architecture, there are several modules which play an important role in the proper working of the banking assistant. Firstly, there is a user interface which provides a way for a user to access the functions provided by the assistant. Here the user can enter his / her query and submit to the assistant for processing.

On submitting a query, first the query goes to the *Tokenizer*. The tokenizer is responsible for generating useful tokens out of the given query. This is usually done by splitting the query sentence into words using delimiters like space, or comma or semi-colon, etc.

The tokens generated from Tokenizer are given as input to a *Normalizer*. A normalizer does the necessary pre-processing on the tokens which includes spell correction of words (using Damerau-Levenshtein distance [7]), expanding of acronyms and abbreviations (by looking into a database of common acronyms and abbreviations) and conversion of tokens to standard format (e.g. all characters in lowercase).

The normalized words are given to a *Parts of Speech (POS) Tagger*, which will assign some meaningful labels to the words. This is well understood with an example. Suppose the sentence is:

"The bank provides 8% interest on fixed deposit."

After assigning appropriate labels, we get the output of POS Tagger as,

"The/DT bank/NN provides/VBZ 8/CD %/NN interest/NN on/IN fixed/VBN deposit/NN ./."

In the above output, DT, NN, etc. are called tags or labels. Here DT stands for Determiner, NN stands for a noun and so on. This information is important as it gives important grammatical properties (tags) which can be used by other modules in the architecture. POS Tagger can be implemented using Hidden Markov Models [8] [9] using Viterbi's algorithm [10].

Next the labelled or tagged output is given to a *Phrase Structure PCFG Parser*. A parser is used to check whether a language follows a pre-defined syntax. In case of natural language, it's very difficult to define a standard grammar as there will be large number of syntactic rules. Thus, some limited number of rules will be used and probability will be used as a measure to determine the best parse for a sentence. Thus, the grammar must be a Probabilistic Context-free Grammar (PCFG). A probabilistic context-free grammar *G* can be defined by the quintuple, *G = (M, T, R, S, P)* where *M* is the set of non-terminal symbols (like NP, VP, NN, etc.), *T* is the set of terminal symbols, *R* is the set of production rules, *S* is the start symbol and *P* is the set of probabilities on production rules [11].

The probabilistic version of Cocke-Younger-Kasami (PCYK) algorithm [12] is well suited for the parser. The output of the parser will be a parse tree. From the example that we used for POS Tagger, the parse tree obtained will be:

```
(ROOT
  (S
    (NP (DT The) (NN bank))
    (VP (VBZ provides)
      (NP
        (ADJP (CD 8) (NN %))
        (NN interest))
      (PP (IN on)
        (NP (VBN fixed) (NN deposit))))
    (. .)))
```

The parse tree obtained from parser is then given to a dependency parser which will convert the phrase structure rules into dependencies. The *Dependency Parser from Stanford's API* [13] is preferred for that. The output will be the dependencies between various words in the sentence. Considering the example from before,

det(bank-2, The-1)
nsubj(provides-3, bank-2)
root(ROOT-0, provides-3)
compound(%-5, 8-4)
amod(interest-6, %-5)
dobj(provides-3, interest-6)
case(deposit-9, on-7)
amod(deposit-9, fixed-8)
nmod(provides-3, deposit-9)

Here, *det(bank-2, The-1)* indicates "*bank*" is dependent on "*The*" by the relation "*det*" i.e. determiner and so on.

These dependencies help in relation extraction by following some hand-written rules for extracting useful relational information from the dependencies. Some typical examples of such relations are quantity (which gives quantity of something; usually obtained from dependency *nummod*), characteristics (like color; obtained from dependency *amod*), main subject in the sentence (obtained from dependency *nsubj*), etc. Considering the previous example sentence, from dependencies we can find relations as mentioned below:

SUBJECT = bank
OBJECT = interest
CHARACTERISTIC = fixed, deposit
ACTION =provides
QUANTITY = 8%, interest

After obtaining the relations, analysis is made to categorize the obtained query in the form of relations into an appropriate type. The types are Account related and General queries. This step is necessary as Account related information should be confidential and for ensuring this, authentication (two-factor) is done so that only the legitimate users access such information.

After this step, the relations are forwarded to a *Relation Matcher* which tries to match the query relation with the relations in a database to find any matching answers. The database consists of relations mapped to their respective natural language sentences. This matching is done using relation scoring as shown in Algorithm 1. The output obtained from the matcher will be the sentence corresponding to the relation with highest score. Based on the relation output, the natural language sentence corresponding to that relation will be obtained and given as output to the user, as the response from the system.

Algorithm 1: Relation matching

1. Let $< A_i, R_i, B_i >$ be a set of relations for $i = 1\ to\ k$ in the database.
2. Let $< P_j, S_j, Q_j >$ be a set of input query relations for $j = 1\ to\ m$
3. Initialize $score = 0$
4. $for\ i = 1\ to\ k$
   $for\ j = 1\ to\ m$
   $if\ A_i = P_j\ and\ R_i = S_j\ and\ B_i = Q_j, then$
   $Add\ 10\ to\ score$

$else\ if\ A_i = P_j\ and\ R_i = S_j, then$
$Add\ 5\ to\ score$
$else\ if\ R_i = S_j\ and\ B_i = Q_j, then$
$Add\ 10\ to\ score$
$else\ if\ A_i = P_j\ or\ B_i = Q_j, then$
$Add\ 2\ to\ score$
$endfor$
$endfor$

5. $return\ score$
6. end

4.3 Architecture using Rule-based approach

The architecture of the system is shown in Fig. 2. System Modules [17] are:

1. Customizable Bot engine.
2. Users questions analytics
3. AIML Interpreter

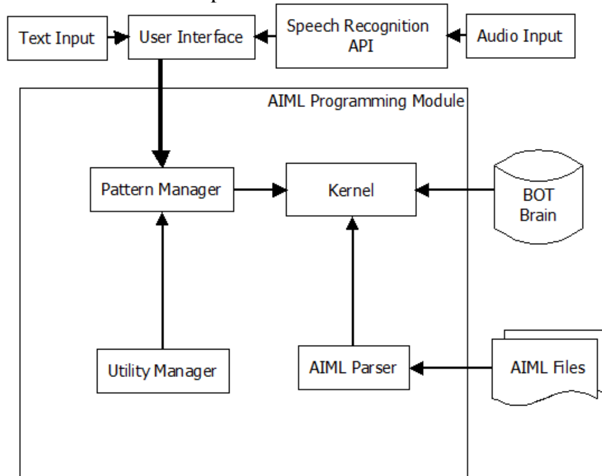

Fig. 2: Architecture of Rule-based approach

4.3.1. AIML Language: Syntax and Semantics

The purpose of AML language is to make the task of dialog modeling easy. Moreover, it is a XML-based markup language and it is tag-based. Tags are identifiers that are responsible to make code snippets and insert commands in the Chatbot. AIML defines a data object class called AIML objects, which is responsible for modeling patterns of conversation. Technically, AIML objects are language tags and each tag corresponds to a language command. The general form of an AIML object/command/ tag has the following structure:

<command> ListOfParameters </command>

An AIML command consists of a start tag (), a closing tag () and a text (ListOfParameters) that contain the command's parameter list. Each line is read, interpreted and executed by software known as AIML interpreter. It is basically a basic unit of dialogue, formed by user input patterns and chatbot responses. These basic units are known as categories, and the set of all categories makes the chatbot Knowledge Base. Among the AIML objects, the following tags are worth citing: category, pattern and template. The category tag defines a unit of knowledge/dialogue of the Knowledge Base. The pattern tag defines a possible user input, and the template tag sets the chatbot response for a certain user input.

The AIML format is as follows:

```
<aiml>
  < topic name=" the topic" >
    <category>
       <pattern>PATTERN…(Your question)</pattern>
       <template>Template…(Answer to the
question)</template>
    </category>
    ..
  </topic>
</aiml>
```

Now an AIML command can be presented using different tags just to increase the scalability of asking queries and also to improve the simplicity of retrieving the answers to the queries.

Also there are certain advanced tags such as, <system> and <javascript> tags interface with other languages; <that> tag stores last response; <topic> tag groups categories together; <srai> tag allows recursion and symbolic reduction meaning it helps to reduce complex grammatical patterns with simple pattern(s); <star/> tag functions the same as a * wildcard [18]. <sr/> is an abbreviation for <srai><star/></srai>. There is also another unique tag know as <sraix> tag. The *sraix element* allows a bot to call categories that exist within another bot, and return response as if it was its own. This enables the creation of many bots, each with a specific purpose that may connect with each other to form a sort of bot network.

For example, we have <that> tag as:

```
 <category
     <pattern>What about increase in rate of interest </pattern>
     <template>Do you like increase in rate of
interest</template>
</category>
<category>
     <pattern>YES</pattern>
     <that>What about increase in rate of interest</that>
     <template>Nice, I like an hike as well. Good for
customers right ?!.</template>
</category>
```

Similarly, for <think> tag we have:

```
<category>
     <pattern>My address is *  . I want a new cheque book
delivered at home </pattern>
     <template>
        Hello!  Alright ! <think><set address = "addr_name">
<star/></set></think>
     </template>
  </category>
<category>
     <pattern>Bye. </pattern>
     <template>
        Bye. Cheque book will be delivered at   <get name =
"addr_name"/>
     </template>
  </category>
```

<random> tag which gives random responses from a list of templates can be used as:

```
  <category>
       <pattern>HELLO *</pattern>
       <template>
```

```
        <random>
            <li> Hello! </li>
            <li> Hi! Nice to meet you! </li>
            <li> Hi! I hope you are doing great! </li>
            <li> Hi! Ssup ! Hope everythings fine ! </li>
            <li> Hello! Wish u a great day ahead today </li>
        </random>
    </template>
</category>
```

Learning in AIML is a bit tedious. Two learning tags that can be used are as follows:

- **Learn Tag:** One can add new set of Queries-Solutions for the bot to answer next time. The new set of Queries-Solutions can be made available while applying updates.(For e.g.: new Banking terminologies).
- **Prefer Tag:** One can improve Solutions for a Query. This will be done after obtaining feedbacks from users to improvise the solutions.

4.3.2 Pattern Matching Algorithm

First Step: Normalization is applied for each input, removing all punctuations, split in two or more sentences and converted to uppercase.

Second Step: AIML interpreter then tries to match word by word the longest pattern match. This is expected to be the better approach. In the proposed work this algorithm can be implemented using two approaches/methods.

Program AB: In program AB two main components are required: (a) AIML file as interpreter between Database and UI, (b) Excel '.csv' file as Database. In this approach, the user interface is Command Line run using '.bat' file and it is completely data driven**.** Basically in such approaches what the Bot is supposed to do is, once the user asks a Query then, the same Query is referred from a list of Query-Solution pairs in AIML Files which are present in the format of Pattern-Template Pairs and these AIML Files have a corresponding csv files (of each AIML file) as Databases to retrieve Solutions to Queries and then it's presented to the User.

Program O: Program O is an implementation of an AIML Chatbot written in PHP. In this, patterns are stored in a database table and instead of matching the pattern to an input, a regular expression is constructed for each input and the database is then searched for pattern strings that match this regular expression. Here just a back-end Server is required.

If the above two approaches are compared then Program O is faster and simpler to implement than Program AB, because in Program AB creation of both AIML files and CSV files is required whereas Program O requires to add data in the backend which is more simpler and faster than adding each Query-Solution pair in CSV files.

System may offer advantages like easy Deployment, high customization, context awareness, and free of cost.

5. CONCLUSION AND FUTURE WORK

Thus, architectures for the banking assistant have been proposed which uses various NLP techniques and Pattern matching techniques for processing the user's queries.

The architecture can still be improved by following a 3-tier architecture [14] where the user interface and some pre-processing is done on the presentation server (first tier), the main processing is done on the application server (second tier) and all databases are on the database server (third tier). In case the entire system is written on just one platform which uses OOP methodology like Java, then the architecture can be distributed with the client and server communicating with each other by means of objects via Remote Method Invocation (RMI) [15]. Alternatively, web service could be implemented [3] which will support multiple clients simultaneously.

There are various issues which have to be considered like dealing with users asking repeated questions intelligently, co-reference resolution, and providing appropriate feedbacks to users in special cases like when user is not asking anything for a long time or when user is continuously asking questions (This can be done by using probabilistic analysis to determine which question will user ask next and by using a timer to determine for how much time the user hasn't asked any question).

A comparative study between two approaches discussed (NLP-based and rule-based) should be done and must be implemented and tested.

6. REFERENCES

[1] "News and opinion: RBS," [Online]. Available: http://www.rbs.com/news/2016/march/rbs-installs-advanced-human-ai-to-help-staff-answer-customer-que.html. [Accessed 1 March 2017].

[2] "Homepage," Kasisto, [Online]. Available: http://kasisto.com/. [Accessed 1 March 2017].

[3] S. J. du Preez, M. Lall and S. Sinha, "An intelligent web-based voice chat bot," in EUROCON, 2009.

[4] "Artificial Linguistic Internet Computer Entity: Wikipedia," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Artificial_Linguistic_Internet_Computer_Entity. [Accessed 4 March 2017].

[5] "AIML: Wikipedia," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/AIML. [Accessed 4 March 2017].

[6] "IBM Watson: Wikipedia," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Watson_(computer)#Software. [Accessed 5 March 2017].

[7] "Damerau-Levenshtein Distance: Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance. [Accessed 6 March 2017].

[8] L. R. Rabiner and B. H. Juang, "A Introduction to Hidden Markov Models," IEEE ASSP Magazine, pp. 4-16, January 1986.

[9] G. Neubig, "NLP Programming Tutorial 5 - Parts of Speech Tagging with Hidden Markov Models," [Online]. Available: http://www.phontron.com/slides/nlp-programming-en-04-hmm.pdf. [Accessed 7 March 2017].

[10] "Viterbi algorithm: Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Viterbi_algorithm. [Accessed 7 March 2017].

[11] "Stochastic Context-free grammar: Wikipedia," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Stochastic_context-free_grammar#Formal_definition. [Accessed 7 March 2017].

[12] G. Neubig, "NLP Programming Tutorial 8 - Phrase Structure Parsing," [Online]. Available: http://www.phontron.com/slides/nlp-programming-en-10-parsing.pdf. [Accessed 7 March 2017].

[13] B. MacCartney, C. D. Manning and M.-C. de Marneffe, "Generating Typed Dependency Parses from Phrase Structure Parses," in LREC, 2006.

[14] "Multitier Architecture: Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Multitier_architecture. [Accessed 7 March 2017].

[15] "Java remote Method Invocation: Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Java_remote_method_invocation. [Accessed 7 March 2017].

[16] Om Komawar, Prasad Thakar, Rohit Shetty, Akshay Bartakke and Manisha Desai,"An Internet Relay Chat Bot using AIML," International Journal of Science and Research (IJSR), Volume 4 Issue 10, October 2015.

[17] Imran Ahmed and Shikha Singh, "AIML Based Voice Enabled Artificial Intelligent Chatterbot," International Journal of u- and e- Service, Science and Technology, pp. 375-384, 2015.

[18] Aniket Dole, Hrushikesh Sansare, Ritesh Harekar, Sprooha Athalye, "Intelligent Chat Bot for Banking System", International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), Volume 4, Issue 5(2), September – October 2015.

[19] " Enhancing AIML Bots using Semantic Web Technologies,"[Online]. Available: http://conferences.idealliance.org/extreme/html/2007/Freese01/EML2007Freese01.html. [Accessed 7 April 2017].

[20] " The Anatomy of A.L.I.C.E."[Online]. Available: http://www.alicebot.org/anatomy.html. [Accessed 7 April 2017].

[21] "ARTIFICIAL INTELLIGENCE MARKUP LANGUAGE: A BRIEF TUTORIAL,"[Online]. Available: https://www.noexperiencenecessarybook.com/5eKql/aimltutorial-aircse-final.html