Intelligent Chat Agent in JESS

Ahmad Ibrahim, Ho-Jin Choi
School of Engineering, Information and Communications University
{ibrahim, hjchoi}@icu.ac.kr

Abstract

An intelligent chat agent is a computer application that can have conversation with the user. This paper gives an overview of design and implementation of such a system in Java Expert System Shell (Jess) and have compared some of the limitations by using two different implementation approaches.

1. Introduction*

The Turing Test is a method for determine the computer (machine) capability of thinking (reasoning) as humans. It was named after Alan Turing who proposed "Turing Test" in his paper “Computing machinery and intelligence”. In the test a participant engages in a natural language conversation with other parties, a human and a machine and if he is unable to distinguish the machine then the machine is said to pass the test. Currently, computer based chat agents are used to simulate the Turing test [1]. In this paper, we have described a similar system build using the JESS (Java Expert System Shell) [7]. However, our research focused on reasoning aspect of such system and not on the complex natural language syntax and semantics.

2. Approach for problem-solving

The intelligent reasoning requires three engines which include sentence matching , keyword matching and one word matching engine and their purpose is to match the user input with the stored vocabulary in the form of sentence and word patterns [1] [6]. Such system also requires features like repetition detection and priority feature. Repetition feature generates different outputs if the user types same input while priority feature give certain sentences and keywords preferences over the other. These features are considered since the user may input same sentence over and over to reveal the system identity while the priority feature enables the system to reason efficiently in case two or more keywords are detected in the input.

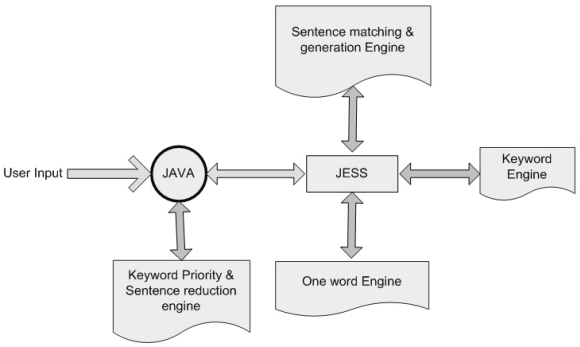
The user input was analyzed by decomposing the input into several parts. The decomposition was done based on the decomposition rules in the vocabulary list. For the input sentence {"It seems that you hate me"} decomposed keywords will be { (1) It seems that (2) you (3) hate (4) me} which will be generated based on the decomposition rules like these {(0 you 0 me),(0 you 1 me),(0 you 0),(0 you 1)} in the vocabulary list. The "0" and "1" are the wild cards representing infinity and only one word respectively. i.e. (0 you 1 me) means that infinite number of words can precede "you" while number of word in between "you" and "me" is equal to one. For the above sentence only rule2 (0 you 1 me) will succeed.

The system output is generated after any decomposition rule is matched with the input sentences. The two types of output generated include response in the form of fix sentences or sentence generated based on the input. For the first case, the output is already defined as shown in Table-1.

<Table1> Predefine output for matched sentences

Input	Output
hello	hello, good to see you
how are you	I am fine
……..	…….

For the second case i.e. Sentence Generation and Re-assembly, the output is generated based on the input decomposed sentence. For the given decomposed sentence {(1) It seems that (2) you (3) hate (4) me}, the corresponding reassembly rule can be {What makes you think I (3) you}. (3) in the reassembly rule will be replaced by the 3rd component of the input , so the output would be { What makes you think I hate you }.



(Fig 1) Overall architecture of application

3. Discussions on implementation

The system was build using two approaches i.e using the Jess Forward Chaining Technique and using Jess Search technique. On the front end Java while on the backend Jess was used (as shown in Fig-1). The tool used for Java was

* This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement).

Eclipse 3.2.1 while the Jess version 7.0 was used [7].

A rank chart was established containing part of vocabulary [5] along with their rank value to implement priority feature. The sentences were given highest value while one-word were given lowest ranking (as shown in Table-2). The similar or related sentences and keywords were grouped under a theme like the sentence at number (24) & (25) represents the SAD Theme. Sentence and keywords under the same theme have the same rank.

<Table 2> Rank chart. Corresponding to each keyword, list of wild cards are written as Rules Applied.

	No.	*Keywords*	*Ranking*	*Rules Applied*
One Word	**1**	i	10	1
	2	yes	40	0,1
	….	……	….	…
Keyword	**5**	i don't	43	0,1
	6	was you	44	0,1
	…	……	..	…
	21	name is	98	0,1
Sentence	**22**	how are you	99	0,1
	….	…….	….	….
	24	i am sick	100	0,1
	25	i hate myself	100	0,1

Using the Jess forward chaining rule, the keyword “I was” was defined as

```
(defrule User_reply_i-was_1
        (declare (salience 47)) (not (key-i-was 12))
        ?f1<-(or    (a i was $?X)(a $?Y i was $?X)
                    (a ?Y i was $?X))
=>
(retract ?f1)(printout t "=>Were you really" $?X crlf)
(assert (key-i-was 12)) )
```

The salience operator of Jess was used to enforce the rank chart within the rules. The repetition detection feature was implemented by asserting a rule in the Jess working memory each time a known word is detected. Some default (generic) sentences can be generated if no rule is applicable. Rules for one word are same as keywords. The output differs as it is generic. One-word represents that the system knows the topic but doesn’t know the context. If the user input {"Yes my brother took position in class"} then the output can be {"You seem to be quite positive"} based on the one-word "Yes".

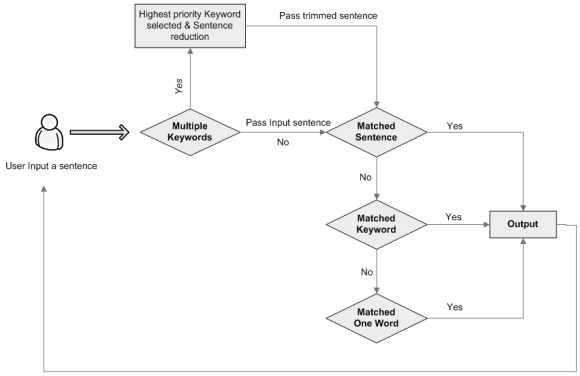
Using Jess Backward Search, part of sentence matching was achieved but the rules were written differently. The input from the users was searched (in the working memory) & the corresponding output was generated accordingly. The matching rules were inserted as Jess facts i.e.

(deffacts data (key "i am sick" sep "oops, i feel sorry").

The execution order of the system (for forward chaining) is shown in Fig-2 while a working system is shown in Fig-3.

4. Conclusion & remaining work

The efficient Rete algorithm of Jess makes the response time of two techniques hardly noticeable. The difference arises in implementation, updation and re-usability of engine.



(Fig 2) Execution order (forward chaining)

```
************   AI Chat Agent Project ***************


*********  Forward Search Engine ***********
Greetings.
Hello to you!
My name is Kim Park , but you can call me Park if you like...
What is your name ...
my name is Lee
=>Please to meet you Mr.lee
hello how are you friend you will be fine soon
=>I am fine
I was laughing at him
=>Were you really  (laughing at him)
I was laughing at him
=>Perhaps I already know you were (laughing at him)
I was thinking that can you smile and look into the camera
=>You believe I can (smile and look into the camera) , don't you ?
yes thats true
=>You seem to be quite positive
```

(Fig 3) Snapshot of the System.

The forward chaining was easy to implement while search was difficult to implement (only sentence matching was implemented without wild cards). In the updation of vocabulary database, new rule has to be defined to add a new sentence in the forward chaining while it was easy in case of search technique as new sentences can be loaded from a vocabulary text file. The reasoning engine in the forward chaining is not so re-usable since matching rules are domain specific while search technique gives the flexibility that once a generic engine for all (sentences, keywords and one-words) is built then the system can be deployed in any scenario since the matching rules are generic in nature and vocabulary of nay domain can be loaded on the run time.

The system can be enhanced further by adding machine learning concept and also by implementing & comparing the current system results with Jess backward chaining results.

References

[1]Joseph Weizenbaum : ELIZA—a computer program for the study of natural language communication between man and machine, Vol 9,Communications of the ACM (1966) 36 - 45

[2]Douglas Walter J. Chubb : Knowledge engineering problems during expert system development ,Vol 15,ACM SIGSIM Simulation Digest archive (1984) 5 - 9

[3]Jason L Hutchens: How to pass Turing Test by cheating , ACM Technical Report,TR97-05 (1997)

[4]Saygin, A. P. et al (2000) Turing Test-50 Years Later, 2001 Kluwer Academic Publishers

[5]Eliza Vocabulary [www.chayden.net/eliza/instructions.txt]

[6] How to build your chatbot [www.codeproject.com/useritems/robomatic.asp]

[7] Ernest Friedman-Hill ,Jess in Action (2003).