SEEKER: A Conversational Agent as a Natural Language Interface to a Relational Database

Emma Victoria Smith[1], Keeley Crockett[1], Annabel Latham[1], Fiona Buckingham[1] and Mohammed Kaleem [1]

***Abstract*—Managers of companies are typically not SQL (Structured Query Language) experts but require information 24/7. Therefore, a growing need for Natural Language Interfaces to Databases (NLIDs) has been identified, with a vast amount of research being undertaken in the area. The existing approaches to NLIDs present many weaknesses including the inability to deal with grammatical mistakes in user input, the inability to communicate with the user to correct mistakes and the inability to allow refinement of query results. This paper proposes a system, SEEKER, which uses a Conversational Agent (CA) as the Natural Language Interface (NLI) in a NLID. The CA is used to capture key words in the user's utterance. Once these key words have been identified, the most appropriate SQL template is selected by the expert system using rule based reasoning. The identified variables are mapped to the SQL template in order to create an SQL query. SEEKER allows for refinement of query results. SEEKER was evaluated in terms of user satisfaction and task completion. The results of the evaluation were promising.**

***Index Terms*—Conversational Agents, Natural Language Interface, Databases**

I. Introduction

Databases are commonly used but require detailed knowledge of a database query language such as SQL. For this reason a manager could not simply click a button to retrieve information from a database. Instead, the manager would have to formally request the required information, which could take time. As the amount of data increases it is essential that information can be retrieved on demand.

A solution to this problem is the use of a NLID. NLIDs have been a popular research area since the late sixties and early seventies [1]. A NLID is a system that allows the user to enter requests in natural language in order to retrieve information from a database [1]. If a NLID is used, the user does not have to perform queries over a database in a database language. Consequently, the user does not need to concern themself with learning a database language, as they can type the query they want to perform in their natural language, for example English. Many different approaches have been taken to creating NLIDs [1]-[9]. These existing approaches present weaknesses, including the fact that the systems do not cope with user input that has incorrect grammar and cannot interact with the user to rectify mistakes and correct unclear input. Furthermore, existing approaches do not allow for the results of one query to be refined further.

This paper introduces SEEKER, a NLID that uses a CA as the NLI. The key features of SEEKER include the ability to enter natural language to retrieve information from a database, the ability to refine or continue with a query's results and the ability to rectify or log mistakes. This addresses a number of weaknesses of previous approaches to NLIDs such as the inability to log errors [4], [5], the inability to overcome incorrect results [3]-[7], and the inability to allow refinement on a query's results [8], [9]. An evaluation was conducted on SEEKER using a set of metrics [10], which were measured by a usability questionnaire and log files produced by the system. The results of this evaluation were positive and suggest that further work in this area would be justified.

This paper is organised as follows: Section II outlines the related work; Section III describes SEEKER's architecture and components; Section IV explains SEEKER's refinement functionality; Section V summarises the evaluation performed along with the results and Section VI includes conclusions and further work.

II. Related Work

A. Natural Language Interfaces to Databases

NLIDs allow the user to enter input in their natural language and then map this user input to an SQL query and execute it. Four main approaches to NLIDs have been established, with each one using a particular system architecture [1], [2]. These approaches are pattern matching systems, syntax-based systems, semantic grammar systems and intermediate representation languages.

Pattern matching systems map user input to the database. The advantage of using a pattern matching approach, compared to other approaches, is its simplicity, as parsing is not involved and systems using this approach usually manage to produce a reasonable answer [1]. However, this is dependent on the domain and this approach has also been criticised for misunderstanding the users input and therefore producing incorrect results [1]. Syntax-based systems, semantic grammar systems and intermediate representation languages all involve some level of parsing [1] [3], [4], [5].

Syntax-based systems parse user input, producing its parse tree. This parse tree is then mapped to an expression in a database query language. Also, a grammar is used to

Manuscript received February, 2014; revised June, 2014.
[1] School of Computing, Maths and Digital Technology, The Manchester Metropolitan University, Chester, Street, Manchester, M1 5GD, UK (corresponding author phone: +44 161 247 1497; corresponding author email k.crockett@mmu.ac.uk)

describe the syntactic structures that the users input could involve. A syntax-based system uses the grammar to create the parse tree representing the syntactic structure of the input. An example of a system that uses this syntax-based approach is LUNAR [3]. Syntax-based systems find it difficult to deal with ambiguous user input as it can lead to more than one parse tree for that user input and they can produce different results [2].

Semantic grammar systems are similar to syntax-based systems, however, the categories in the grammar do not represent syntactic concepts. Instead, they represent semantic concepts. Semantic grammar systems focus on simplifying the parse tree as much as possible because the system can then reflect the semantic representation better. Examples of systems using this approach include PLANES and LADDER [1]. Semantic grammar systems are difficult to port to other domains [2].

An intermediate representation language is a language that is used between a higher-level language and a lower-level language. Systems that use intermediate representation languages take user input and express it as an intermediate logical query in an intermediate representation language, such as LQL (Logical Query Language). The logical query is then transformed into a database language query and evaluated over the database. An advantage to this approach is the ability of porting the system to different databases. The more current NLIDs use this intermediate representation language approach [1], [2].

This intermediate representation language approach was used in creating MASQUE/SQL [4]. MASQUE/SQL only supports Ingres for UNIX. Androutsopoulos et al [4] claim that the system can be easily ported to any DBMS that supports C with embedded dynamic SQL. Conversely, there is no evidence that this has been tested. The MASQUE/SQL system contains a domain-editor, which uses an is-a hierarchy, to help the user describe entity types of the database domain. Therefore, it is expected that the user puts forward the words that they expect to appear in the natural language questions. Also, the user is expected to define the meaning of each of these words in terms of a logic predicate. Even though the system offers the domain-editor to support the user in these tasks, they will still need to have some expert knowledge. Androutsopoulos et al [4] address some limitations of MASQUE/SQL themselves. One of these limitations is due to when an SQL query fails. The system cannot locate which part of the query caused the system to fail and therefore the message it produces to the user is not particularly informative. Another limitation is due to the mapping of each logic predicate to a database table, view or query, as it can sometimes cause redundant joins in the SQL queries.

The intermediate representation language approach to creating NLIDs was also used to create a NLID called Edite [5], which uses SQL. Reis et al [5] point out some possible limitations of their system. One limitation that is discussed is the fact that the system can only take in questions, not statements. Another limitation considered is that there is no way for the system to log any failures that are encountered. If the system could log failures then these logs could be used to improve the system.

There are a small number of NLIDs that do not fall into the four main approaches. These NLIDs include PRECISE [6] and Nihalani el al's [7] NLID. PRECISE used an approach containing several components; a lexicon, a tokenizer, a matcher, a parser plug-in, a query generator and an equivalence checker. Nihalani et al [7] created a NLID by splitting the system architecture into two modules; a pre-processor and a run time processor. The pre-processor uses the database metadata to create rules. The run time processor parses the user input and tries to match these words with those in the domain dictionary.

The approaches to creating NLIDs, previously discussed, cannot always effectively deal with user input containing incorrect grammar [8]. These approaches do not offer effective functionality for communication with the user, which would allow ambiguous input or incorrect results to be overcome [8], [9]. Furthermore, a number of the systems that use these approaches do not offer an effective way for errors, such as the system failing or producing incorrect results, to be logged [4], [5]. This means the user is not very well informed and as errors are not recorded they cannot be corrected.

B. Conversational Agents

A Conversational Agent (CA) allows a user to converse with a machine in natural language. This has been a goal within Artificial Intelligence since the Turing Test [11]. A CA is expected to play the role of a human expert in some domain. For example, CAs have been useful in applications for bullying and harassment [12] and student debt advice [13]. Typically, a CA uses a pattern matching based approach to converse with the user. Pattern matching identifies key words in the user's utterance and tries to match them with a pattern. This results in a response being displayed to the user.

The term CA covers a wide range of types. One type of CA is a Chatterbot, which converses with a human while trying to keep the conversation going for as long as possible [10]. Chatterbots, which have been developed to undertake the Turing Test as part of the Loebner Prize [14] have been said to use trickery to convince users they are human [10], [15]. An example of trickery employed is taking the human utterance and rephrasing it to use in the next response to the user. Therefore, Chatterbots are restricted in their ability to hold a meaningful conversation. However, techniques employed by Chatterbots, such as pattern matching, can be adopted to create a more robust CA for use in applications. One of the earliest Chatterbots to be developed was Eliza [16], a psychotherapist based CA. The trickery employed by Chatterbots worked well for Eliza due to the nature of psychiatric interviews. However, other domains may require more conversation then just rephrasing of user input.

ALICE (Artificial Linguistic Internet Computer Entity) has been used many times for the creation of CAs, including three winning occasions of the Loebner Prize in the years 2000, 2001 and 2004 [17]. ALICE is a general purpose CA, of the Chatterbot type, created using AIML (Artificial Intelligence Markup Language). ALICE is advantageous as AIML is simple. However, it has limited functionality, as it does not allow the capturing of variables from the conversation.

ISBN: 978-988-19252-7-5 (revised on 24 June 2014)
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Another type of CA is a Goal Oriented CA (GO-CA), which engages in conversation with the user to achieve a specific goal. Therefore, the agent has a purpose in the conversation unlike a Chatterbot. Usually, a GO-CA is developed using pattern matching techniques [10] [18], but more recently semantic sentence similarity measures have been used to overcome issues related with scripting and maintenance [18].

C. Conversational Agent-Natural Language Interface to Databases

Research has already been undertaken into Conversational Agent-Natural Language Interfaces to Databases (CA-NLID) [8], [9]. Pudner et al [8] created a NLID with an existing CA, InfoChat, from Convagent Ltd [13]. The NLID also contained an expert system, control module, and relational database. The majority of modules in this system were unable to be ported to other domains.

Owda et al [9] took a similar approach to create C-BIRD (Conversation-Based Interfaces to Relational Databases). The components C-BIRD includes are a knowledge tree, a CA, SQL query templates, tree for dynamic generated queries, relational database metadata and database annotation, a relational database, information extraction module, response generation and a conversation manager.

These existing CA-NLID systems have some limitations. Neither of the CA-NLID systems [8], [9] allow for refinement to be performed on query results. Refinement could include adding columns to query results or narrowing down query results by removing either column(s) or row(s). Refinement could also provide functionality for results of one query to be used in a new query, for instance in a comparison. An example of a comparison would be asking for a company's total sales in a particular year and then asking the system to compare this with the total sales of a different year. Similarly, the CA-NLID systems [8], [9] do not provide functionality for a query to be modified e.g. adding, removing or changing of an attribute. Additionally, the existing systems [8], [9] do not provide a facility for the user to receive an explanation of the results. This would be useful, as the user will understand how the system has interpreted their input, in order to generate a set of results.

Clearly there are advantages of using a CA as an NLID such as, the handling of grammatically incorrect user input, the production of log files of each conversation and the ability to engage in conversation with a user to clarify ambiguous input or to rectify incorrect results. However, the problems of Pudner et al's [8] CA-NLID and the C-BIRD system [9] still need to be addressed. The SEEKER system proposed in this paper will overcome these problems by allowing refinement to be performed on a query's results. Refinement will allow additional columns to be added to results, as well as allowing results to be narrowed down by reducing the columns or the rows of the query results. Additionally, the system will allow for conversation to take place on whether the results were correct.

III. Overview of the Seeker System

This section describes SEEKER. The main features of the system are:

- Engage in conversation with a user in natural language, in order to extract information to build a query, to retrieve the information required by the user
- Execute the query over the database and display the results to the user, without them having to view any SQL
- Allowing conversation with the user around the topic of refinement and therefore allowing refinement to be performed on query results
- Allowing conversation with the user around whether the results were correct
- Creating log files recording each conversation and any module errors
- Answering a set amount of Frequently Asked Questions (FAQs)
- Keeping the conversation on the topic of the domain
- Making sure the language the user uses is not inappropriate, e.g. swearing, and informing them if it is
- Allowing conversation around a set amount of unrelated topics, giving the CA personality, while directing the user back to the topic of the domain
- Overcoming weaknesses of previous NLIDs as it allows errors to be logged, allows incorrect results and ambiguous input to be overcome and allows refinement on a query's results.

A. Architecture

The modules within SEEKER are shown in Fig 1.

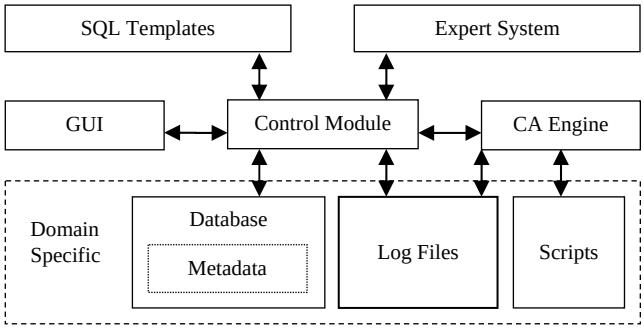

Fig. 1. SEEKER System Architecture

An overview of the workflow SEEKER follows is outlined below.

1. User enters an utterance into the Graphical User Interface (GUI) and the GUI passes this utterance to the control module
2. Control module passes the user utterance to the CA engine
3. CA engine processes the user input, using the scripts. It produces an appropriate response and retrieves any variables which were set in the scripts
4. Response and variables are passed to the control module
5. Variables are passed to the expert system for processing
6. Expert system determines the SQL template needed and passes the name of this template to the control module
7. Control module uses this SQL template name to access the correct SQL template
8. SQL templates pass back the requested SQL template to the control module

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

9. Control module maps the variables to the template and executes this query over the database, retrieving the results
10. Control module passes the response from the CA engine and/or the results of the SQL query to the GUI, were they are displayed to the user

Each module will now be briefly described.

B. Database

An Oracle 11g relational database was used with an existing travel company domain [19]. This travel company sells tickets to attractions all over the world. The domain was adapted for use with SEEKER, by reducing the amount of tables and attributes used, in order to test proof of concept on a set of queries that users would typically ask the travel company. This allowed the database to resemble a real life situation but focus on certain aspects. Scenarios were designed to guide users into asking questions that would formulate SQL queries covered in SEEKER.

C. Conversational Agent Engine

A CA engine is required to provide functionality to capture attributes within the scripts. An existing CA engine, Convagent Ltd, was chosen for use with SEEKER [13]. This CA engine was chosen as it was a GO-CA and provided the best functionality in terms of architecture and scripting language abilities for a CA-NLID system from the CA engines considered. The CA within SEEKER is called Emma.

D. SQL Templates

A questionnaire was designed and developed to capture common SQL queries and natural language. Results of this analysis questionnaire provided many natural language questions that could be asked to the travel company, and their corresponding SQL queries. This allowed a number of SQL templates to be chosen for use with SEEKER. There were six essential SQL templates implemented within SEEKER, all are SQL *select* queries, which are part of the Data Manipulation Language (DML). An example of one of the SQL templates used in SEEKER is shown in Fig 2.

```
select att_one, att_three
from table_one, table_two
where table_one.join_att_one = table_two.join_att_two
and att_two like '%attribute_two_val%';
```

Fig. 2. Example of an SQL Template used within SEEKER

This template, Fig 2, allows two attributes to be displayed in the results window, while joining two tables together. A third attribute it used to narrow down the results produced to the user. This template was chosen as it was essential for allowing SEEKER to perform refinement.

E. Scripts

The scripts for the CA were implemented in the PatternScript scripting language provided by [13]. Contexts were used to allow conversation about each table in the database. Contexts consisted of a number of rules with patterns associated with the user input. Rules were included to answer several FAQs where the database could not provide the answer. A personality layer was also included to give the CA personality. This layer answered a number of questions about the CA, for example its favourite football team, while trying to direct them back to the travel company domain. Strategies were implemented to deal with utterances unrelated to the domain and utterances that are not tolerated e.g. swearing. These strategies involved politely warning the user about their utterance twice and then if the user repeated the offence a third time SEEKER closes.

F. Expert System

The role of the expert system is to determine which SQL template is the most appropriate for a particular user utterance. It contains a set of generic rules and takes in variables set in the scripts. Each rule contains an SQL template name, which is returned when the rule is fired. The expert system creates a rule base with all possible rules that could be fired for a particular utterance. It then executes the rule base by sorting and determining which rule to fire based on the rules priority number.

G. Control Module

The control module performs many tasks which include:

- Accessing other modules in the architecture (Fig 1) allowing for any module failures or errors to be detected and dealt with
- Checking the database metadata as it contains facts about the domain, i.e. on the tables, attributes, attribute values and relationships between tables
- Checking the syntax of the variables to be mapped
- Mapping the variables set in the scripts to the SQL template that has been retrieved
- Refining a previous queries results
- Writing any module errors or failures to the log files

H. Log Files

The CA engine provides functionality for log files to be created containing the conversation between the user and SEEKER. The log files record utterances input by the user and responses from the CA. Additionally, they include the rules in the scripts which contain patterns that match the utterance, and the rules that fired. Therefore, this can be used in the evaluation, through the recording of specific metrics, as it is clear whether the correct rule fired or not.

I. Graphical User Interface

The GUI allows the user to enter their utterance, displays the conversation between them and SEEKER and displays the query results.

IV. Query Refinement

SEEKER can perform refinement on query results for the user. The refinement allows the user to narrow down results or find out further information about the results. When the user is displayed query results, and refinement is possible, the user is asked if they would like to refine. If the user does want to refine then they are asked how they want to refine. An example of a dialogue with SEEKER, using refinement, is shown in Table 1.

ISBN: 978-988-19252-7-5 (revised on 24 June 2014)
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE I
EXAMPLE CONVERSATION WITH REFINEMENT

	Utterance
Emma:	Hello! My name is Emma. I am here to help you with information on the attractions we sell tickets for. How can I help you today?
User:	Can you tell me all the attractions you sell tickets for
Emma:	I am showing you the attractions we sell tickets for! Shall I help you refine them? Please let me know by entering yes or no
User:	Yes
Emma:	How would you like to refine or continue with these results?
User:	I just want the attractions in paris please
Emma:	Here are the attractions in Paris! Shall I help you refine them? Please let me know by entering yes or no
User:	Yes
Emma:	How would you like to refine or continue with these results?
User:	Now I want the prices of those attractions
Emma:	The prices are now being displayed! Would you like my help to refine them? Please let me know by entering yes or no
User:	Yes
Emma:	How would you like to refine or continue with these results?
User:	Now I want the disneyland paris ticket types too
Emma:	Here are the ticket types for Disneyland. Was this what you were looking for? Let me know by entering yes or no
User:	Yes thanks
Emma:	Brilliant! Feel free to ask me something else, or if you want to exit just type 'bye'
User:	Bye
Emma:	It was nice chatting to you, goodbye!

As the example conversation in Table 1 shows, the user can use the refinement functionality to narrow down results as well as add additional attributes to the results. Results can be narrowed down by reducing either the attributes or the rows. Refinement can also be used to rectify incorrect results or misunderstandings. An example of this involves the user asking for all attractions in Paris but SEEKER retrieving all attractions instead. This misunderstanding can be solved by using refinement to narrow attractions by the city of Paris.

V. Evaluation

A. Evaluation Methodology

The evaluation methodology undertaken on SEEKER was to evaluate two components; task completion and user satisfaction. Each component was evaluated using existing metrics [10]. Three types of metrics were chosen for use in the evaluation; subjective, objective and query-oriented. User satisfaction was evaluated using eleven of the metrics, which were measured using a usability questionnaire completed by participants. Metrics measured by the usability questionnaire were all subjective metrics except one query-oriented metric (No. 8 in Table 2). Task completion was evaluated by the other ten metrics, which were measured using log files that SEEKER produced during conversations with participants. Metrics measured using log files were a combination of objective metrics and query-oriented metrics (No. 18-20 in Table 3). The evaluation methodology involved several methods because evaluation of both CAs and NLIDs is a difficult topic that is still being researched [10].

B. Experimental Methodology

The experiment involved ten participants, with one participant at a time using SEEKER and undertaking the evaluation. Each participant was given a set of scenarios and the usability questionnaire to read, and verbally given a short explanation of SEEKER. They used SEEKER on a laptop and engaged in a conversation with SEEKER following the scenarios. SEEKER produced log files for each conversation. While the participants were completing the scenarios they were unobtrusively observed allowing any interesting observations to be recorded.

C. Discussion and Results

Metrics measured by the usability questionnaire are shown in Table 2 and metrics measured by the log files are displayed in Table 3. All metrics are shown with their corresponding percentages summarising how well each metric was achieved.

TABLE II
METRICS MEASURED BY USABILITY QUESTIONNAIRE

No.	Metric	%
1	The agents understanding of the user utterances	72%
2	How natural the agent's behavior seemed	79%
3	User satisfaction	80%
4	Ease of correcting misunderstandings	84%
5	Ease of user understanding the agent	84%
6	Whether the agent behaved as expected	85%
7	Whether the user would use again or prefer human service	86%
8	The users query retrieved the correct results	88%
9	Ease of use	92%
10	How well the CA controlled the conversation	93%
11	Friendliness of the agent	96%

TABLE III
METRICS MEASURED BY LOG FILES

No.	Metric	%
12	Number of times the system crashed	0%
13	Number of times the CA misunderstood the user	4%
14	Number of times the CA gave an incorrect response (not including the misunderstandings)	10%
15	Number of times the CA gave an incorrect response (including misunderstandings)	14%
16	Number of times the CA gave a correct response	86%
17	Number of times the CA answered questions correctly	88%
18	Number of times the system produced incorrect results	4%
19	Number of times the system failed to return any results	24%
20	Number of times the system produced the correct results	72%
21	Number of times the results could be refined	100%

Metric results produced for SEEKER were positive, however, participant evaluations allowed for any issues SEEKER had to be identified. One issue identified was some participants using 'short form' utterances, which SEEKER did not accept, as more of a conversation was expected. Examples of 'short form' utterances include *"by paris"* and *"attractions"*. This resulted in the user having to rephrase the utterance to receive query results. Although 'short form' utterances were not expected as they are not typical in a conversation with a human, they could be expected when the user tries to refine results. SEEKER did not always recognise utterances that it should have recognised and therefore closed after the user entered the utterance three times. Furthermore, when the user entered an

ISBN: 978-988-19252-7-5 (revised on 24 June 2014) WCE 2014
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

utterance that was not recognised, while refining the results, SEEKER asked the user to rephrase the utterance and try again. When the user rephrased the utterance SEEKER no longer performed refinement and treated the utterance as a new query.

D. Key Findings

All problems encountered were due to one module, the scripts, as they allowed the CA to misunderstand and not recognise several utterances. These types of problems were expected as natural language is a vast area, and they can easily be corrected. The participants' got use to utterances SEEKER accepted, and did not mind occasionally correcting the CA's misunderstandings, highlighted by the metric *ease of correcting misunderstandings* receiving 84%. This is helpful as there are vast amounts of ways people can enter utterances that mean the same, and it is difficult to script them all. Although these scripting problems occasionally prevented results from being retrieved, the success rate for the correct results returned was still high. If the scripts were improved the percentage for the correct results would improve because all problems identified with SEEKER were in the scripts module. All other modules worked correctly and therefore they had a 100% success rate.

VI. Conclusions and Further Work

This paper introduced SEEKER, a system that contains a CA as the NLI to a relational database. SEEKER uses the CA to capture key words in a user's utterance and translates this into an SQL query using an expert system and a set of SQL templates. SEEKER allows for query results to be refined and allows for errors to be recorded in log files. Misunderstandings or incorrect results can be overcome with either the utterance being rephrased or with refinement. Natural language is a difficult area as one utterance can be said in many ways. This means 100% accuracy is difficult to achieve. SEEKER received good results in the evaluation for all metrics used. There was a 72% success rate for the number of times SEEKER produced the correct results. All modules, except the scripts, could be ported to another domain. Results of SEEKER are positive and justify further work in this area. Developing SEEKER further would need a larger sample of participants for analysis and evaluation, with extensive testing. SEEKER would benefit from the scripts being extended not only to correct the problems found in the evaluation but to allow more conversation. It would be useful to extend the amount of SQL templates that SEEKER includes, as a larger variety of user utterances would be accepted. SEEKER allows for refinement of query results but this does not include comparisons. SEEKER could be extended to include comparisons, as part of refinement. This would involve extending the SQL templates and the scripts, as it could use the refinement functionality that is already in place. SEEKER could benefit from an explanation facility, where it could explain results it has produced. This could help to overcome any misunderstandings and incorrect results, along with the use of refinement.

Acknowledgment

The authors would like to thank Convagent Ltd for the use of their CA engine within SEEKER.

References

[1] I. Androutsopoulos, G. Ritchie, and P. Thanisch, "Natural Language Interfaces to Databases – An Introduction" *Journal of Natural Language Engineering*, vol. 1, 1995, pp. 29-81

[2] N. Nihalani, S. Silakari, and M. Motwani, "Natural Language Interface for Database: A Brief Review" *International Journal of Computer Science*, vol. 8, 2011, pp.600-608

[3] W. Woods, R. Kaplan, and B. Webber, "The Lunar Sciences Natural Language Information System", Final Report, Technical Report 2378, Bolt Beranek and Newman Inc., 1972.

[4] I. Androutsopoulos, G. Ritchie, and P. Thanisch, "MASQUE/SQL – An Efficient and Portable Natural Language Query Interface for Relational Databases" in *Proceedings of the Sixth International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems.* Edinburgh, 1993.

[5] P. Reis, J. Matias, and N. Mamede, "A Natural Language Interface to Databases: A New Dimension for an Old Approach" in *Proceedings of the Fourth International Conference on Information and Communication Technology in Tourism.* Edinburgh, Scotland, 1997.

[6] A. Popsecu, O. Etzioni, and H. Kautz, "Towards a Theory of Natural Language Interfaces to Databases" in *Proceedings of the Eighth International Conference on Intelligent User Interfaces.* Miami, Florida, July 2003, pp. 149-157

[7] N. Nihalani, M. Motwani, and S. Silakari, "Natural Language Interface to Database using Semantic Matching" *International Journal of Computer Applications*, 31, 2011, pp. 29-34

[8] K. Pudner, K. Crockett, and Z. Bandar, "An Intelligent Conversational Agent Approach to Extracting Queries from Natural Language" in *Proceedings of the World Congress on Engineering.* London, 2-4 July 2007

[9] M. Owda, Z. Bandar, and K. Crockett, "*Information Extraction for SQL Query Generation in the Conversation-Based Interfaces to Relational Databases (C-BIRD)"* Manchester: Manchester Metropolitan University, 2011

[10] J. O'shea, K. Bandar, and K. Crockett, "Chapter 8: Systems Engineering and Conversational Agents" in A. Tolk, and L. Jain, (eds.). *Intelligent-Based Systems Engineering*. Berlin: Springer-Verlag, 2011, pp. 201-232

[11] A. Turing, 'Computing Machinery and Intelligence.' *Mind*, 49, 1950, pp.433-460

[12] A. Latham, K. Crockett, and Z. Bandar, *"A Conversational Expert System Supporting Bullying and Harassment Policies"* Manchester: Manchester Metropolitan University, 2010

[13] Convagent Ltd. (2005) *Conversational Agents.* [Online] Available: http://www.convagent.com/default.htm

[14] The Loebner Prize Contest. (1991) *Homepage of the Loebner Prize in Artificial Intelligence.* [Online] Available: http://www.loebner.net/Prizef/loebner-prize.html

[15] S. Shieber, "Lessons from a Restricted Turing Test" *Communications of the ACM*, 37, 1994, pp. 70-78

[16] J. Weizenbaum, "ELIZA – A Computer Program for the Study of Natural Language Communication between Man and Machine" *Communications of the ACM*, 9, 1996, pp. 36-45

[17] A.L.I.C.E. (no date) *A.L.I.C.E. Artificial Intelligence Foundation.* [Online] Available: http://alice.pandorabots.com/

[18] K. O'Shea, Z. Bandar and K. Crockett, "A Novel Approach for Constructing Conversational Agents using Sentence Similarity Measures" in *IEANG*, London, 2-4 July 2008

[19] K. Crockett, S. Morris, P. Rob, and C. Coronel, *Database Principles: Fundamentals of Design, Implementations and Management.* 2nd ed., Cengage Learning, 2013

Date of Paper modification: 24th June 2014
Brief description of the changes: Mr. Mohammed Kaleem is added as a co-author.