Asia Pacific Journal of Contemporary Education and Communication Technology
ISBN (eBook): 978 0 9943656 8 2 | ISSN : 2205-6181
Year: 2017 , Volume: 3, Issue: 1

SMART INTELLIGENT EXPERT SYSTEM TO TROUBLESHOOT ORA-ERRORS

K.S. Amaraweera[a], K.M.C.B Kendaragama[b], W.B.U. Dayananda[c],
K.D.K. Shamantha[d], U. Samaratunge[e]
[abcde] Sri Lanka Institute of Information Technology, Malabe, Sri Lanka
Corresponding email: amaraweerasaranga@gmail.com

Abstract

Oracle database products are used by database administrators, system engineers, developers and testers in the software industry for different purposes. The Oracle database domain has over 60,000 ORA-Errors and yet, no proper system solution is found to troubleshoot errors in a few steps. According to the evidence found, it takes a considerable amount of time and effort to fix an ORA-Error manually. Therefore, this project the 'Smart Intelligent Expert to Troubleshoot ORA-ERRORs' is a software product which attempts to address the above issues. The system runs on Oracle Database 12C product in Linux platform. The Automatic Error Correcting component of the system resolves ORA-Errors upon the user request with the help of domain specific Ontology. The Question and Answering component helps users to clarify further issues regarding the Oracle database domain using Natural Language Understanding and Natural Language Generation. The ultimate goal of this research is to help Oracle database users, especially database administrators to troubleshoot ORA-Errors generated in the Oracle database without mucA effort.

Keywords: ORA-Errors, Linux Platform, Oracle Database 12C, Natural Language Understanding, Natural Language Generation

1. Introduction and purpose

Oracle database is an Object-Relational Database Management System produced and marketed by Oracle Corporation. Oracle works on most of the platforms and the literature reveals that oracle database products are more competitive against open source relational database management systems. The ORA-Errors in different product versions can vary from each other due to the features released with each product versions. The release of new features with the late versions causes the number of ORA-Errors to increase. Fixing small issues which occur in the database, again and again, may be a considerable inconvenience for Database Administrators since there are so many tasks to handle in a database. It is really helpful if there is someone who fixes that kind of small errors for them. When an error is thrown in the Oracle database, less-experienced users might have to ask help from experts in the domain or else they have to search through the internet. It is not guaranteed that the answers which are found on the internet are 100% reliable. Then, the error has to be solved manually. The solution can be either a file configuration, a command line execution or change in databases.

This manual method of solving ORA-Errors takes a lot of time and wastes significant effort. Even the database administrators find it hard to fix errors of huge databases. Therefore, the need of a proper mechanism to resolve these errors without wasting much effort and time is

quite essential. Automating a solution to the issue using an expert system creates a faster and cost-effective solution and it will reduce the downtime of the users to resolve Oracle database errors. The project 'Smart Intelligent Expert to ORA-ERROR Troubleshoot' intends to address all above issues. 'Smart Intelligent Expert to ORA-ERROR Troubleshoot' is a software that runs on Linux platform along with the Oracle database product. When an ORA-ERROR is thrown in Oracle database product, the user can request to fix the solution. Then, the system will take relevant actions to solve the error by the Automatic Error Correcting component. When the error fixing is beyond its capabilities and it will interact with the user through Question Answering process and will enable the user to clarify errors by providing necessary solutions. This will minimize the interaction with the database system files. Currently, such implemented system cannot be found in Oracle troubleshooting domain. This research will be the first attempt to troubleshoot ORA-Errors.

2. State of Art

A. Oracle Error Domain and Oracle Database Product

Oracle started releasing products since 1978, starting from Oracle Version 1 and up to Oracle 12c. Table 1 clearly depicts the increased number of ORA-Errors from version to version from 1998.

Table 1: Oracle database versions with the number of ORA-Errors

Database version	Error Number
Oracle 8i (1998)	ORA-0000 to ORA-30999
Oracle 10g (2004)	ORA-0000 to ORA-40322
Oracle 11g (2007)	ORA-0000 to ORA-62001
Oracle 12c (2013)	ORA-0000 to ORA-65535

The users who encounter ORA-Errors are frequently database administrators and software engineers. They can be experts in the domain or less-experienced people. In the real world scenario, it takes less time to fix an error for an expert than an inexperienced person. But, considering the severity level of handling huge databases, even a fraction of second becomes very important. Fixing some errors can take lots of effort due to some reasons. In the worst case, there are some ORA-Errors which cannot be fixed without Oracle Supporters. Therefore, Oracle database administrators have a huge responsibility on their hands to maintain databases without causing problems.

Most of the troubleshooting systems described in Table 2 are only capable of providing solution tips for a certain problem. The implemented system is capable of resolving selected subset of errors itself using Automatic Error Correcting System. Solution tips are provided only if the error cannot be fixed by the system. This process will be so much easier for users than receiving solution tips for even simple and easily fixable errors.

Table 2: Similar products comparison

	SharePlex	ADDM	AutonomusDB	Proposed Product
Intelligent?	yes	no	Yes	yes
Troubleshoot problems Automatically?	no	no	No	yes

Answer to Users' Questions	no	no	Yes	yes
Secure remote access	yes	no	No	no
Free	no	yes	No	yes

B. Troubleshooting Related Expert Systems with Knowledge Base

In March 2015, another expert system was proposed by International Journal of Engineering Research and Management (IJERM) for troubleshooting the errors in the Windows 7 Operating System where the system extracts a rule for each error, this rule is used so that the system can take recommendations from the database and exhibit them to the user for completing the repairing process (Moraes, Salgado & Tedesco, 2009). The diagnosis process is completed by a neural networking process and it identifies the cause of the issue and takes necessary actions to resolve the problem. Also, the user can communicate with the system by entering the symptoms through a question-based setup and then the expert system provides recommendations of how to fix the error and the causes for the issue.

In 2013, an expert system was suggested by a group of researchers in Cross River University, Nigeria which can troubleshoot PC faults and assist PC end-users in dealing with their PC problems and also assist PC technicians in accurate diagnosis of PC fault by providing a systematic and stepwise analysis of failure, possible cause(s) of the failure and offer maintenance recommendations (Sylvester & Adesola, 2013). The knowledge base is defined with rule-based reasoning and decision-making methods and in order to identify the faults, it examines the situation given by the user input and fixes the problem or provides recommendations to fix it.

In 2007, a research group from National Technical University of Athens, Greece proposed another expert system for power transformer troubleshooting and maintenance which assists users to fix failures in the power transformers. The system consists of a knowledge base and a set of production rules applied through forward and backward chaining process. The Knowledge Base is constructed in a tree form with three main branches corresponding to Preventive, Predictive and Corrective Maintenance (Troubleshooting) (Kaminaris, Moronis, Kolliopoulos, Sakarellos, Kontargyri & Kampanaros, 2007). The expert system gives suggestions and step by step instructions to the user by a set of questions and also continuously checks the failure of the power transformer and the causes for the failures. The domain knowledge is represented by a tree structure and the data are being processed by that pre-defined structure so that the system can find possible reasons.

C. Question and Answering Systems

The first conversational agent 'Eliza' which was developed by Joseph (1996) was capable of tricking some people and letting them think that they are communicating with a real person. It had a simple design with a script which matches keywords and generates answers. However, in the Artificial Intelligent world conversational dialog is a challenging aspect (Kenny & Parsons, 2011).

Loebner (1991) started Loebner Prize Contest to find out 'the most thinking computer'. This competition has been held annually. The first winner was 'Eliza' and since then there have been many winners like Mitsuku, chatbot Rose, Lisa, Izar, and many more (The Society for the Study of Artificial Intelligence and Simulation of Behaviour, 2014).

'Edgar' is another good example for a chat agent which uses a knowledge base to analyse the input and generate a meaningful output. It uses a technique called Edgar's Machine Technique to achieve good results (Pereira, 2013).

'Just.Chat' is a platform for processing information to be used in chatbots (Pereira & Coheur, 2013). They have used a knowledge base to create the corpus instead of writing it from scratch. The text analysis and understanding is conducted by passing the chat corpus through different filters and then classifying them into personal or impersonal questions. The filters are responsible for different tasks like domain, the personality of the user etc. They have used another filter to understand the question by looking for synonyms in WordNet. In that way, the system understands the question. The answer is extracted from AIML files included in the knowledge base (Pereira, 2013).

SPARQL queries support to retrieve data from the ontology. Natural Language Interfaces to Databases explains how data interact with the structure of the database. Each component of the NLU takes an input and provides an output for the next component making the final result an SPARQL query (Djahantighi, Norouzifard, Davarpanah & Shenassa, 2008).

Linked Open Data Question Answering system clarified that SPARQL queries are dominant in querying linked data through a semantic web. So, it provides an interface to access open RDF data by understanding natural language and generating SPARQL queries (Kim & Cohen, 2013).
Python NLTK is the most significant, leading library for implementing programs to work with natural language data. It contains all structural components using in natural language understanding. Many NLP frameworks are built under or inspired by NLTK.

Earlier, most of the times, people tend to use scripts to analyze and understand (Natural Language Understanding) the user's question. At present, scripts have replaced the ontologies and the answer is formatted in a meaningful way using Natural Language Generation (Chakrabarti, 2014).

According to Reiter and Dale's 'Consensus' architecture, Natural Language Generating systems can be built using pipeline architecture following three steps. Although this architecture is acceptable, it has a few drawbacks like fail to create more accurate detailed functional descriptions and many more (Mellish, Scott, Cahill, Paiva, Evans & Reape, 2006).

3. Methodology

The system mainly consists of three components.

i. Ontology-Based Model Process for Oracle Error Troubleshooting
ii. Automatic Error Correcting System
iii. Question Answering System

The Ontology-Based Model Process represents the knowledge for the identified problem domain by establishing specified set of rules to interact with the inference engine in the expert system. Automatic Error Correcting System is responsible for fixing ORA-Errors occurred in the Oracle database product while the Question Answering System answers user's questions by extracting answers from the knowledge base. Figure 1 shows the High-level design of the system.

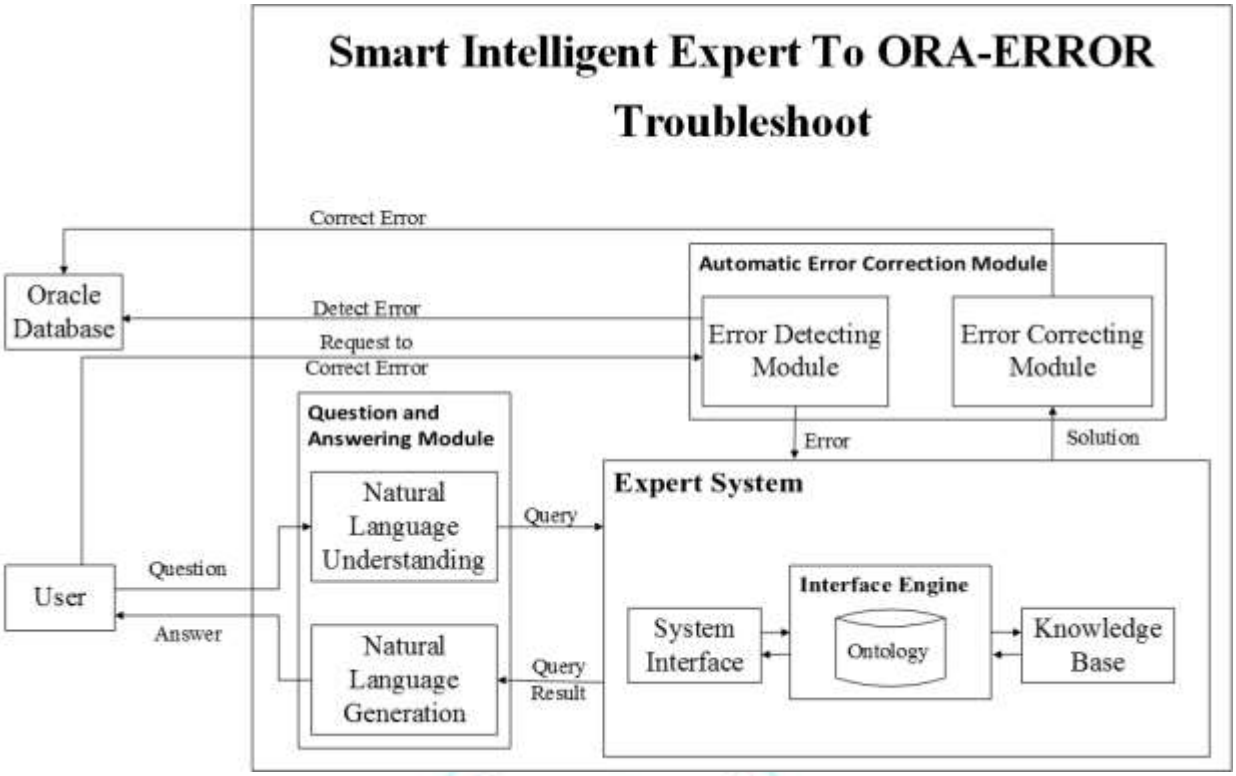

Figure 1: High-level diagram of the system

D. Ontology Based Model Process

This difficulty can be overcome by a semantic approach as the oracle error domain is quite large and searching solutions for the ORA-Error over the internet takes a lot of time. Using an existing ontology model for the problem takes a considerable amount of effort to understand and apply the suggested system, since the limited set of rules and resources in an existing ontology, constructing our own ontology model is the optimal solution for the problem domain. All knowledge in the Oracle error domain is represented by the constructed ontology by establishing a set of SWRL rules which can shape the behavior and the interconnections of the entities.

The entire structure of the ontology based model process is described in brief. The ontology was constructed using Protege 4.3 framework. The class hierarchy was created according to Oracle error domain. Defining the classes in the hierarchy was created by the top-down development process. That means creating the classes for the general concepts and then specialize them with subclasses. For example, Solutions class has subclasses like Steps and Autofix in order to provide solutions for the generated oracle error.

Table 3: Ontology Class Hierarchy

Class	Description
ORA-ERROR	Has all the information related to Oracle error and this class is mapped with object relationships for automatic error correction, and solution tips.
Solutions	All the solutions as steps and tips for the relevant oracle error is mapped by the Solutions class. It has two sub classes for automatic error correction module and solution tips.
Steps	Solutions as tips are provided by this class and mapped with the relevant oracle error
AutoFix	Have all stored script locations for automatic error troubleshooting process for the relevant Ora-Error

All data related to the ontology are stored as RDF triples. The functional representation of the constructed ontology is represented by SWRL rules and also it is comfortable with RDF data. This sample rule shows the relationship of a given ORA-Error.
ORA-Error(?x), Solutions(?y) -> hasTips(?x, ?y)

A sample rule would be ORA-Error/Solutions/Steps are mapped in with 'hasSteps' object relationship where it limits the data related to automatic troubleshooting process. That means the knowledge extracts only the possible solution tips for a given ORA-Error.

The main characteristic of the constructed ontology is the ability to make decisions, reacting to the specified actions and elements in the expert system. All the elements are stored in an OWL file which is responsible for reusing the domain knowledge and its preferences. In order to react to the inference rules, the ontology uses reasoning process. Using tableau-based decision procedure for the reasoning process will ensure the efficiency of the decision-making in the ontology. HermiT 1.3.8 Reasoner, which comes with Protégé used for this purpose.

In order to omit the overlapping constraints between the entities, the use of set operations were used. The members of a class are specified by using the set operations in Protégé framework. Equivalent and disjoint classes have been declared in order to omit the anomalies between the troubleshooting related processes.

Using the domain knowledge for the troubleshooting and inferring the knowledge activities are done by SPARQL querying process. Apache Jena 3.1.0 framework was used to manipulate the knowledge since Jena processes RDF data with different APIs. SPARQL queries are manipulated with a set of rules and techniques used in Jena framework. When the user mentions the affected Ora-error, it should provide a suitable solution as a result by acquiring the knowledge from the ontology. Querying process bridges the connection with the ontology and the inference engine. After the queries are processed, the inference engine gets the corresponding solution. Therefore, an efficient querying process is essential to the system in order to generate the solutions.

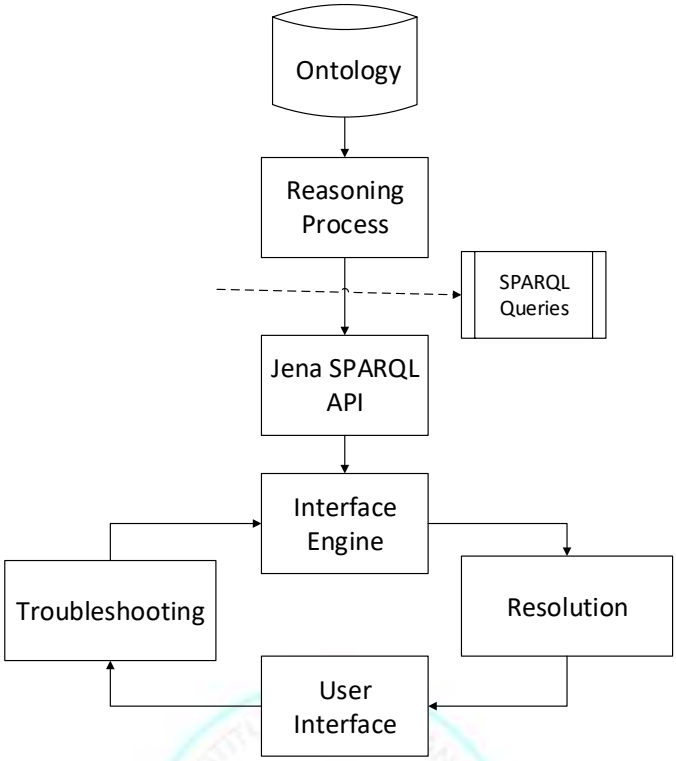

Figure 2: Constructed ontology interacting with SPARQL queries and inference engine

E. Automatic Error Correcting System

The Automatic Error Correcting system is the module in which the Ora-errors are automatically fixed by the system. This module starts functioning when an Ora-error is matched with the given set of solutions in the knowledge base.

After the checking process of an error, the inference engine looks into the knowledge base and if it is fixable, then the automatic error correcting system should follow the step by step instructions given in the knowledge base and execute them. Then, it shows whether the operation is successful or not to the user. In case if the error cannot be fixed automatically by the system, it directs the user to question agent. Furthermore, it generates a status report for all the related troubleshooting processes done by the users.

The Automatic Error Correcting module uses executable bash scripts and shell scripts in order to access the Oracle database. When the specified error is mentioned by the user via the user interface, it connects through the SPARQL query module and checks whether the error is fixable by the system. If it is fixable, the Automatic Error Correcting module looks into the exact script path located in the ontology. Eventually, the script executor module executes the shell script in order to resolve the specified ORA-ERROR. The estimated time can be varied in resolving the ORA-Error according to the severity of the error.

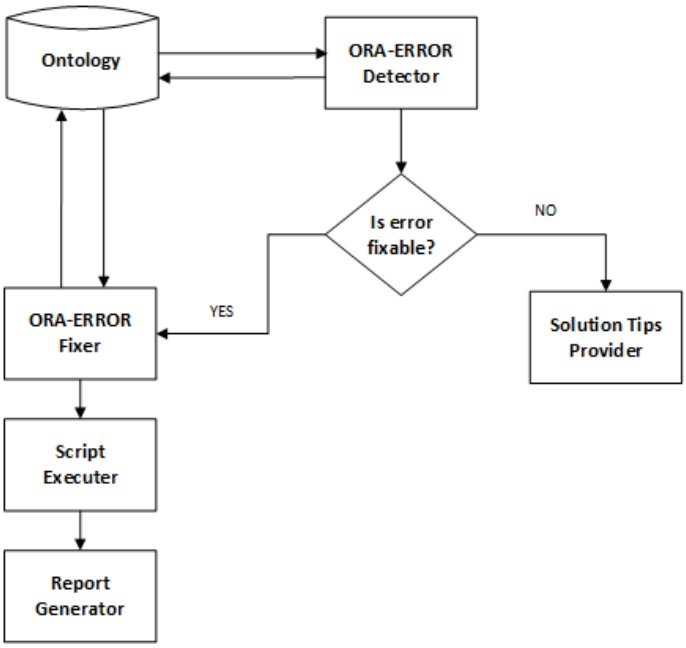

Figure 3: High-level diagram for Automatic Error Correcting component

This process will reduce the downtime of fixing the generated ORA-ERROR so that the users will not have any hassle in searching the solutions for them. Additionally, apart from the Error Correction process, the users can further clarify their issues regarding the generated Oracle error by asking for solution tips and using the question answering system.

F. Question and Answering System

The system will get user questions as text input and parse that to the knowledge base to retrieve correct data. While recognizing entities in the formless text it understands user questions using Natural Language Understanding (NLU).
User input should be in the English language. Spelling mistakes in typed text questions are checked before processing to understand. The system uses the Peter Norvig Proposed Model for spelling correction.

The corrected quest is parsed into predefined structure of the input question as a regular expression. Quepy framework's regular expressions contain POS tags, Lemma, and target keywords. The tokenized question passes through each and every regex question pattern defined inside classes. Interpret method is called after matching the question with regex.
Quepy DSL uses an abstract semantics as a language-independent representation and then it is mapped to a query language. It allows user questions to be converted to an SPARQL query inside interpret method. Returns of the interpreted method are appended with some meta-data of the question in order to understand the input question.

Output SPARQL query goes through ontology interface. SPARQL query results and meta-data from the interpreted method, construct a XML string for Natural Language Generation process.

The Natural Language Generation component transforms the output of the Natural Language Understanding component into a descriptive and more understandable format in English.
This component receives solution for user's question from the Knowledge Base in XML format. The root element takes the name of the class defined in the ontology where the answer is

extracted. The body part represents the answer. The XML key-value pairs are mapped as the subject and object of sentences. This job is done by Data Mapper under the Sentence Planner subcomponent.

The Data Mapper converts the XML formatted answer into an object of a predefined data structure called Paragraph Bean, which includes a root element, a root element name, a question type (What, Where, When or How) and a list of sentence structures. The sentence structures contain Subject, Verb, Object and parameters that decide the grammar rules (ie. the sentence is negative or positive, the number of the subject and tense).

WordNet component is used to identify nouns, verbs, adverbs, adjectives and tense of the Verb when mapping XML elements at 'Data Mapper'. If the value of an element is a noun or a noun phrase, it sets "be" verb for the Verb and the noun or the noun phrase for the Object. If the element value is a verb, the root element is set as the Subject, the key is set for the verb and the Object is the element value. The Wordnet component handles all the operations done with Wordnet Dictionary using JWNL java library. JWNL facilitates many functions like identifying word form (noun, verb, adjective or adverb), returning all the synonyms for a given word and returning the base form of a word.

The Grammar Checker subcomponent checks the list of sentences in Paragraph Bean and makes the changes to the sentence accordingly. This component uses SimpleNLG to set the tense of the sentence, to make subject singular or plural and to convert the sentence into the negative form. The conjunctions are set by combining two objects of adjacent sentences where subject + verb are equal. The same logic was applied to combine sentences considering verb + object. There, the sentence is set to plural after combining two subjects.

In the final step, pronouns are set by checking adjacent sentences where the subject is the root element. Then, it places the pronouns accordingly. If two adjacent sentences are having the same subject, determiners are set to the second sentence.

To set grammar, pronouns, conjunctions and determiners, SimpleNLG 4.3.3 is used.

There is no any proper mechanism to retrieve the adverb relevant to an adjective. This function is used when setting Object of a sentence as in Table IV.

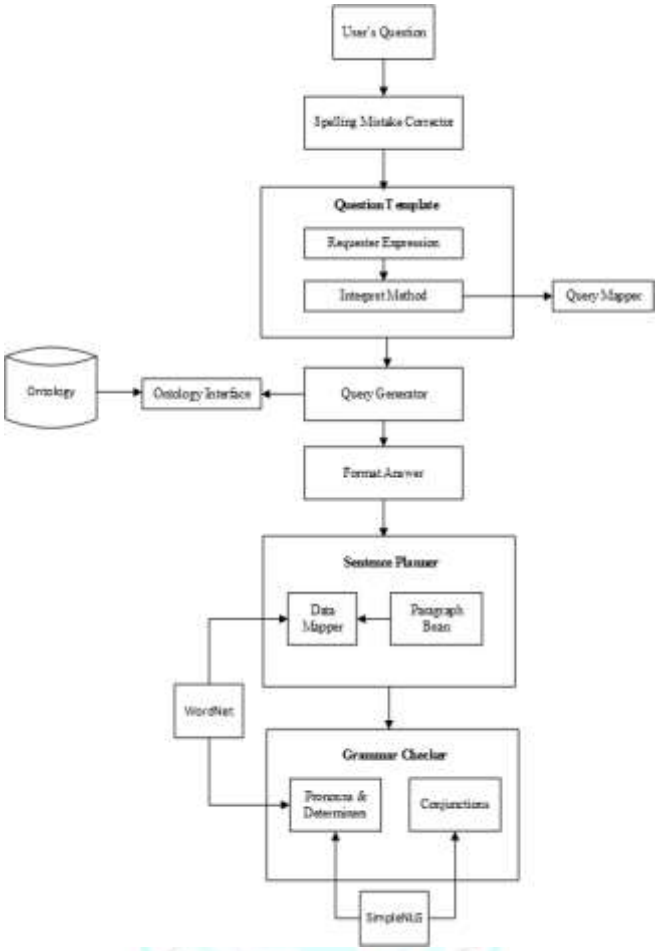

Figure 4: High-level diagram of question and answering component

Table 4: SVO values assigned to SPhraseSpec object

SPhraseSpec attribute	Value assigned by reading the XML
Subject	She
Verb	Ran
Object (Here an adverb is assigned as the object)	quick

When converting Table IV example to a sentence, the object should be converted to an adverb. The output should be "She ran quickly". The conversion mechanism was implemented with the help of WordNet. In the usual scenario, an adverb has a "y" at the end of the word. Therefore, system checks synonyms for "quick" and checks if that is such a case and returns the adverb. If it is a special case, the system returns a suitable adverb by checking all the synonyms.

Up to WordNet 3.0 and SimpleNLG 4.3.3, there aren't any links between nouns and pronouns. Therefore, finding pronouns for a noun should be implemented by the system. Such facility is needed for constructing formatted paragraph in English. Since this is a domain specific question and answering component, most of the times third person personal pronouns were used for domain specific nouns.

Third person personal pronouns can be divided into different categories like number (singular or plural) and gender (masculine or feminine). Before implementing the logic, more than a hundred tests were done on WordNet 3.0 to find out the most commonly used words in the descriptions under synonyms of different nouns. Table V shows commonly used words for similar sets of nouns and the relevant pronoun. These collections are stored in a property file where the key is the pronoun and value is the word collection.

Table 5: Pronoun and relevant WordNet map

Pronoun	Sample set of nouns	Commonly used words to describe relevant nouns (word collection)
He	father, student, waiter…	man, boy, person
She	mother, sister, waitress…	female, woman, girl
There	mountain, school, university…	place, location, area
It	apple, dog, pen,,,	file, error and every noun that does not fall under above categories

When a noun is given to find relevant pronoun, the system reads property file and find if a word in the collection matches with words in the description under each synonym.

4. Results and Discussions

The constructed ontology was tested by using the reasoning process in Apache Jena framework and Protege's inbuilt reasoner, HermiT 1.3.8. In this scenario, basically, the rules were tested out using a set of test cases. The class hierarchy, data members and their relationships between them were tested by running the HermiT reasoner and according to the test results, logical and overlapping constraints in the ontology have been minimized.

In order to test whether the ontology retrieves the correct responses for a specific SPARQL query, Jena Fuseki server and SPARQL queries were used. A sample SPARQL query would be,

The query explains the number of input fields and input field names that are needed for error ID, ORA-00942 for the automatic troubleshooting process.

```
PREFIX ora:<https://localhost/orafixing.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?no_of_inputs ?input_fields_names
WHERE
{
        ?x ora:error_id ?errorID .
        FILTER(?errorID = "ORA00942").
        ?x ora:autoFixes ?autofix.
        ?autofix ora:no_of_inputs ?no_of_inputs.
        ?autofix ora:input_fields ?input_fields_names
}
```

Figure 5: SPARQL query

The sample output is,

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
        <variable name="no_of_inputs"/>
        <variable name="input_fields"/>
 </head>
 <results>
        <result>
        <binding name="no_of_inputs">
        <literal datatype="http://www.w3.org/2001/XMLSchema#int">2</literal>
        </binding>
        <binding name="input_fields">
        <literal>Table Name, User Name</literal>
        </binding>
        </result>
 </results>
</sparql>
```

Figure 6: Output XML for the SPARQL query

The result shows that number 2 of input fields and 'Table Name', 'User Name' fields are needed for troubleshooting ORA-00942 error.

Extract key words from the user question and generates a SPARQL query, according to the extracted key words. For question "what is ora12541?" extracts ORA-error code "ora12541" and question phase type that "what" to generate SPARQL. Figure 7 illustrates output SPARQL that goes through ontology and retrieve data to produce answer.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ora:<https://localhost/orafixing.owl#>
SELECT DISTINCT? property? value WHERE
{
?x0 rdf:type ora:ORA-Error.
?x0 ora:error_id "ORA12541".
?x0 ?property ?value.
}
```

Figure 7: SPARQL query for question "what is ora12541?"

TSV format of the received data from the ontology from SPARQL in Figure 7 shown in Figure 8.

```
?property	?value
<https://localhost/orafixing.owl#error_description>"TNS no listener"
<https://localhost/orafixing.owl#is_fixable_by_the_system>true
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<https://localhost/orafixing.owl#ORA-Error>
<https://localhost/orafixing.owl#error_id>"ORA12541"
<https://localhost/orafixing.owl#hasTips>
<https://localhost/orafixing.owl#Tip-ORA-12541>
<https://localhost/orafixing.owl#need_Oracle_support_to_fix>false
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#NamedIndividual>
<https://localhost/orafixing.owl#severity_level>"MInor"
<https://localhost/orafixing.owl#caused_due_to>"Listener for the source repository has not been started."
<https://localhost/orafixing.owl#autoFixes>
<https://localhost/orafixing.owl#AutoFix-ORA-12541>
<https://localhost/orafixing.owl#hasSteps>
<https://localhost/orafixing.owl#Step-ORA-12541>
```

Figure 8: TVS output from ontology

Extracted data from the TSV is combined with question meta data information and generated xml illustrate in Figure 8.

```
<question_type>WHAT</question_type>
<error>
 <error_description>TNS no listener</error_description>
 <is_fixable_by_the_system>true</is_fixable_by_the_system>
 <error_id>ORA12541</error_id>
 <need_Oracle_support_to_fix>false</need_Oracle_support_to_fix>
 <severity_level>MInor</severity_level>
 <caused_due_to>Listener for the source repository has not been started </caused_due_to>
</error>
```

Figure 9: Formatted XML

The input for the Natural Language Generation component is a string in XML format as in the Figure 9.

The ParagraphBean holds data of a paragraph. Root element of the JSON string is stored as the rootElement, question type as the paragraphClauseType and sentences in a sentenceList where the data type was SPhraseSpec of SimpleNLG library. If there is a name or an ID which describes root element, later it sets root ElementName. Table 6 shows the values to the ParagraphBean class parameters according to the example in Figure 9.

Table 6: Paragraph Bean class variables

rootElement	Error
paragraphClauseType	WHAT
rootElementName	OORA000001

The JSON string is converted into a SPhraseSpec object in SimpleNLG library. The SPhraseSpec object stores information mentioned in Table 6 of a sentence.
JSON element : "is fixable by the system": "false"

Table 7: SPhraseSpec object parameters

Subject	Error
Verb	Is
Object	Fixable by the system
Negative	True
Plural	False
Tense	Present

Table 7 illustrates the logic behind setting subject, verb, object of a sentence. Further, the tense of the sentence was checked and set at the same time. Negation of a sentence were also considered.

Ora000001 is not fixable by the system. The error is caused due to table owner name not specified when logged-in as a non-creator of the table, ora-00942 on table import (imp or impdp), ora-00942 on materialized view refresh. Severity level is minor. They mean 'the table or view does not exist'. They will not need oracle support to fix.

Figure 10: Final output

The completed system is evaluated by a group of database administrators, system engineers, application engineers and Oracle database certified users who use Oracle database on regular basis. The main purpose of this section is to identify the strengths and highlight the areas that need to be further concerned.

Acknowledgement

We would like to take this opportunity to pay our profound gratitude towards the people who supported us in all ups and downs of this research project, especially Mr. Udara Samaratunge for being our examiner and giving us valuable comments and feedbacks in times of need. We also would be in debt to the lecturers who guided us in various different fields and to the staff of Sri Lanka Institute of information Technology for guiding us throughout to achieve the project goal successfully.

Conclusions

The Smart Intelligent Expert to Troubleshoot ORA-ERRORs can troubleshoot generated ORA-Errors with less effort and time, and it provides better solutions for specified set of ORA-Errors.

According to the results obtained during the testing and evaluation phases of the developed application, it showed that Smart Intelligent Expert to Troubleshoot ORA-Errors clearly addresses the mentioned problem domain. In addition, gained results proved that this application is at a considerably higher level of usability and reliability along with the featured components. Further, analysis process has shown that developed application is fully capable of handling the defined set of ORA-Errors.

The smart intelligent expert system is not only capable of fixing Oracle database errors automatically, but also helps the users to clarify further questions by the Question and Answering module. Therefore, the Oracle database users can take the maximum profits and benefits from the developed application.

Since the developed application basically focuses on resolving a selected subset ORA-Errors, the system can be extended furthermore to address the issues in all the ORA-Errors generated in the Oracle database as future work.

The constructed ontology can be expanded for further improvements. The Automatic Error Correcting module can be further enhanced by increasing the number of fixable ORA-Errors. Currently, the Question and Answering system validate questions that users can only ask WH questions. It is better if the system could understand and generate answers for any kind of user question. This will need enhancement in Natural Language Processing component.

References

i. Chakrabarti, C. (2014) "*Artificial Conversations for Chatter Bots Using Knowledge Representation, Learning, and Pragmatics*," 2014.

ii. Djahantighi, S., Norouzifard, M., Davarpanah, H. & H. Shenassa, H. (2008) 'Using natural language processing in order to create SQL queries.' In *International Conference on Computer and Communication Engineering, Kuala Lumpur: IEEE*, 2008, pp. 600–604.

iii. Kaminaris, S.D., Moronis, A.X., Kolliopoulos, N. I., Sakarellos, A. P., Kontargyri, V. T. & Kampanaros, S. D. (2007) 'A new expert system application for Power Transformer Troubleshooting and Maintenance.' In *World Scientific and Engineering Academy and Society: Proc. of 6th Int, Corfu Island, Greece*, *February 16-19, 2007*, pp. 52-57.

iv. Kenny P. & Parsons T. (2011) 'Conversational Agents and Natural Language Interaction.' In Perez-Marin and Diana (Eds.) *Embodied Conversational Virtual Patients*, pp. 254–281.

v. Kim, J. & Cohen, K. (2013) 'Natural Language Query Processing for SPARQL generation - a Prototype System for SNOMEDCT.' In *Intelligent Systems for Molecular Biology, Berlin, Germany, July 20, 2013*, pp. 32–38.

vi. Mellish, C., Scott, D., Cahill, L., Paiva, D., Evans, R. & Reape, M. (2006) 'A Reference Architecture for Natural Language Generation Systems', *Natural Language Engineering*, 12(1), pp. 1–34.

vii. Moraes, A. Salgado & Tedesco, P. (2009) 'AutonomousDB: a Tool for Autonomic Propagation of Schema Updates in Heterogeneous Multi-Database Environment. In *International Conference on Autonomic and Autonomous Systems*, *Valencia, Spain, April 20-25 2009*.

viii. Pereira, M. (2013) *Just.Chat - a platform for processing information to be used in chatbots*.

ix. Sylvester, E. & Adesola, W. (2013) *Design of Computer Fault Diagnosis and Troubleshooting System,* 9, pp. 43-53.

x. The Society for the Study of Artificial Intelligence and Simulation of Behaviour (2014) 'Loebner Prize Notice.' [Online]. Available at: http://www.aisb.org.uk/events/loebner-prize

Accessed: February 20th, 2016.