A Frame-based Approach to Text Generation

Huong Le_Thanh
Faculty of Information Technology
Hanoi University of Technology, Vietnam
1 DaiCoViet Street, Hanoi, Vietnam
huonglt@it-hut.edu.vn

Abstract

This paper is a study on constructing a natural language interface to database, concentrating on generating textual answers. TGEN, a system that generates textual answer from query result tables is presented. The TGEN architecture guarantees its portability across domains. A combination of a frame-based approach and natural language generation techniques in the TGEN provides text fluency and text flexibility. The implementation result shows that this approach is feasible while a deep NLG approach is still far to be reached.

1 Introduction

Database management systems (DBMSs) have been widely used thanks to their efficiency in storing and retrieving data. However, retrieving information from database requires users to compose queries in a query language (e.g., QBE or SQL) or to fill some search criteria in the interface (Liu, 1995; Catarci et al., 1997). In addition, traditional DBMSs are still limited by its capability to generate outputs. They normally dump query results in a table or a pre-defined form without any understanding of the meaning of data.

The research on natural language interface to databases has recently received attention from the research communities (Wang et al., 1999; ELF Software Co., 2001; Torgersson and Falkman, 2002; Hallett et al., 2005; Bertomeu, 2006). The purpose of such a natural language interface is to allow users to compose queries in natural language and to receive responses under the form of short answers. The natural language interface is thus preferred than the traditional interface.

A typical natural language interface has to solve two main tasks: (i) translating a natural language query to a query language; and (ii) generating a natural language answer by using information from a query result table. In this paper, we focus on solving the second task. TGEN – a text generator developed by us - can generate query responses under the form of short answers or summary reports in Vietnamese language.

The remaining sections of this paper are organized as follows: Section 2 introduces the architecture of our proposed Natural Language Interface to Database (NLI2DB), in order to get an overview of the TGEN role in the interface. Section 3 describes the rule set used by the TGEN to generate text. Section 4 presents the major components and the data flow in the TGEN. In Section 5, two examples are given to illustrate the working process of the TGEN. Our implementation discussion and some experimental results are given in Section 6. Finally, Section 7 concludes the paper and proposes possible future work on this approach.

2 The NLI2DB Architecture

The NLI2DB is a system that is integrated with a traditional database management system to provide a natural language interface for querying database and automatically generating answers. An overview of the NLI2DB architecture is shown in Figure 1.

The NLI2DB consists of two main modules:

- A Query Translator (QTRAN) to translate natural language questions to SQL queries.
- A Text Generator (TGEN) to generate answers from query result tables.

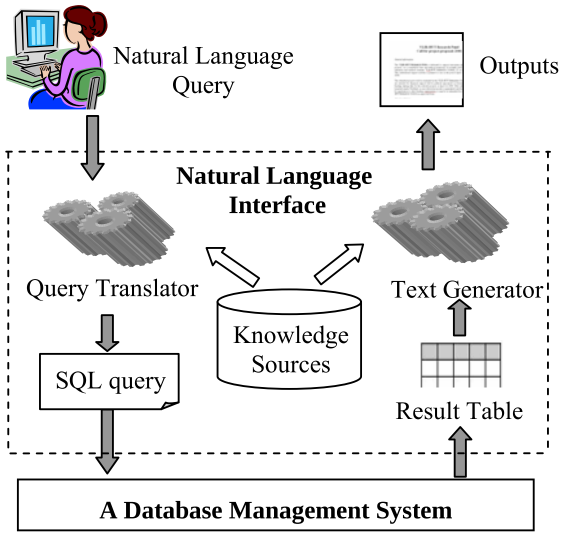

Figure 1. Architecture of the NLI2DB

A natural language question composed by a user is translated into a SQL query by the Query Translator. After that, the Database Management System processes the SQL query and returns the query result in the form of a table. The Text Generator then transforms this result table into a textual answer.

To test the feasibility of the NLI2DB, a specific Database Management System - a student management database – is used. The entity relationship of the database is shown in Figure 2. Knowledge sources (e.g., syntactic rules and thesaurus) are needed in the working processes of the QTRAN and the TGEN.

The remained sections present our main focus of this paper - the implementation of the TGEN. The rule set used in generating answers is introduced first.

3 The Grammar used in the TGEN

The TGEN does not generate free texts, but the texts that are based on predefined frames. These frames are typical structures of answers. For example, the frame for an answer of the *List* type is:

[Noun phrase] [Verb phrase]:
1. [Item_1]
2. . . . . . .
3. [Item_n]

A *List* answer can also be represented by another frame:

[Noun phrase] [Verb phrase] [Item_1], …, [Item_n].

Each label *[. . .]* in the frame is a slot that needs to be filled. The *[Noun phrase]* and the *[Verb phrase]* are generated by using the syntactic structure of the user's question. This problem is analyzed in detail in Section 5. The slots *[Item_1]*, ..., *[Item_n]* are filled in by values from the result table.

In order to create the frame set that is used in generating text, we first identify and categorize question types, then we define frames for each question type. The question types that have been considered by us are:

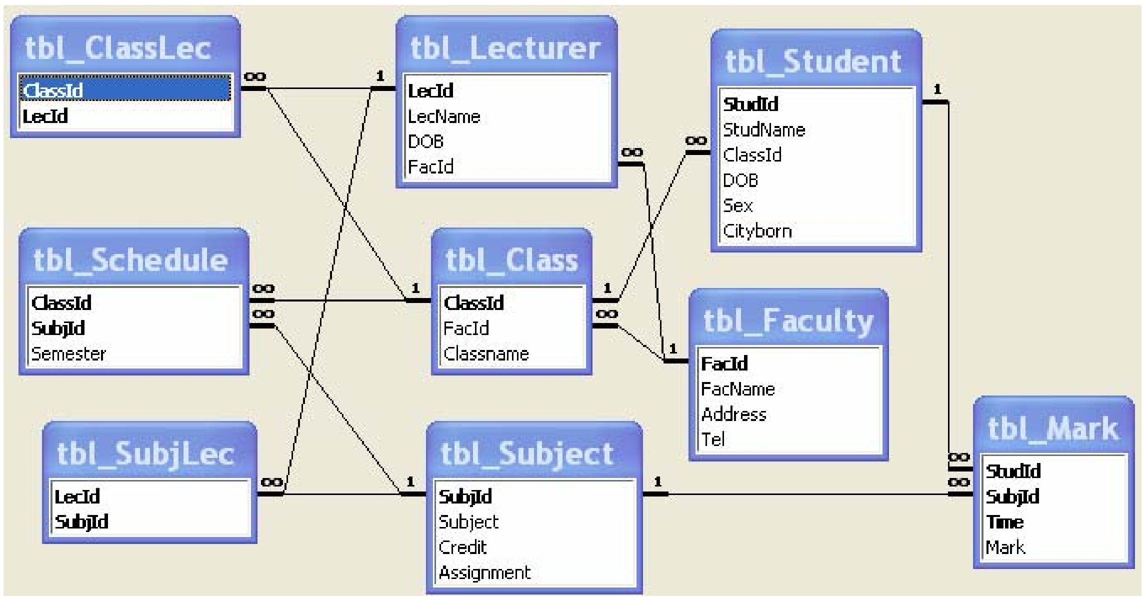

Figure 2 – The entity relationship of the student management database

1. Questions that return a single value (e.g., *Who is the leader of the class BK20 in the academic year 2004-2005?*[1]). This question type is called a *Single_value* question.
2. *List* questions (e.g., *Which subjects did the class BK20 study in Semester 1 last year?*)
3. *Statistical* questions (e.g., *Show us the quality of students in the academic year 2005 – 2006.*)
4. *Comparison* questions (e.g., *Compare the percentage of excellent students of the classes BK20 and BK21.*)
5. *Description* questions (e.g., *Give us information about the student Pham Thanh of the class BK20.*)
6. *Evaluation* questions (e.g., *Evaluate the study progress of the student Nguyen Van Minh.*)

The name of a frame is called by its corresponding question type. For example, the frame of a *List* question is a *List* frame.

The TGEN uses a context-free grammar (CFG) to organize generation rules. Each frame in the TGEN is stored in the right hand side (RHS) of a rule, whose left hand side (LHS) is the frame type (e.g., *List* frame, *Comparison* frame). The set of generation rules used in the TGEN is called a rule set.

The LHS of a generation rule can be any non-terminal symbol, whereas the RHS is a combination of non-terminal symbols and terminal symbols. A terminal symbol is a word or a string that appears in the output text. A non-terminal symbol do not appears in the output. Instead, it has to be expanded by other generation rules or to be replaced by a value in the query result. In order to produce flexible output texts, a non-terminal symbol in the RHS of a rule can also be a frame. We analyze the *Description* frame to illustrate the organization of our rule set.

The answer for a *Description* question is a text that describes relations among attribute values of one or several entities. In order to keep the generality of the frame set, we design frames for each entity of the database. Each frame consists of all attributes of an entity. During the generation process, if some slots in the frame do not have values to fill in, these slots will be removed from the frame. The following rules are applied to relations among attributes of the *tbl_student* entity.

(1) [frame_student] → [studname] ([sex]) [vp_studId]. [studname] [vp_DOB], [vp_cityborn]. [studname] [vp_classId].

(2) [frame_student] → [studname] ([sex]) [vp_studId]. [studname] [vp_classId]. [studname] [vp_DOB], [vp_cityborn].

(3) [vp_studId] → has the student code [studId]
(4) [vp_DOB] → was born on [DOB]
(5) [vp_cityborn] → in [cityborn]
(6) [vp_classId] → is a student of the class [clasId]
(7) [vp_classId] → studies in the class [classId]

In the above rule set, the strings in the square brackets (*[ ]*) are considered as non-terminal symbols; whereas the string that are not in the square brackets are terminal ones.

The rules whose LHS starts with *[frame_* define the structure of a frame. Rules (1) and (2) in the rule set above define two possible structures of the *Description* frame. The non-terminal symbols starting with *[vp_* or *[np_* represent for verb phrases or noun phrases, respectively. The rule set also has an *[s_* symbol, which represents for sentence. The symbols that are in the square bracket and do not start with *[frame_*, *[vp_*, *[np_* and *[s_* are pre-terminal symbols. They represent for entity's attributes and will be replaced by attribute values during the next generating step.

If a *Description* answer describes relations among attribute values of several entities, the system will automatically generate the answer by connecting the frames of these entities through entity's key. The system only select frame's parts that relates to attributes of the result table. During the generating process, the entities' keys are often replaced by their corresponding name attributes (e.g, [*StudId*] is replaced by [*studname*]). This process will be illustrated by Example 1 in Section 5.

[1] For the convenient, all examples are translated from Vietnamese to English in Sections 3,4 and 5. In Section 6 (Implementation Discussion and Experiments), we present our original examples in Vietnamese and translate them into English.

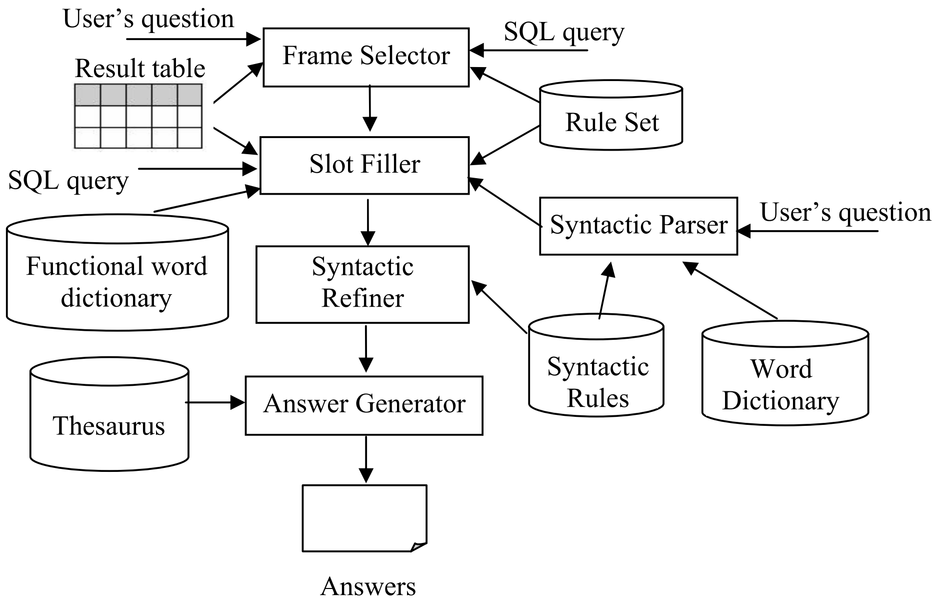

Figure 3. Data flow in the Text Generator

4 The Text Generator system

The major components and the data flow in the TGEN are shown in Figure 3. The Text Generator is divided into four main components: a Frame Selector, a Slot Filler, a Syntactic Refiner, and an Answer Generator.

4.1 A Frame Selector

This component is used to select frames for the answer. Four factors being considered in selecting frames are: keywords in the user's question, the SQL query[2], the shape of the result table, and values in the result table. Each factor will be analyzed in detail below.

Some user's questions contain keywords that signal theirs question types. Examples of the keywords are *Compare*, *List*, and *Evaluate*. If these keywords are found in the user's question, the frames corresponding to the detected question type will be chosen.

The shape of the result table can decide the frame type as well.

- If the result table has only one value, the *Single_value* frame is selected.
- If the result table has several columns and one row, the *Description* frame is chosen. If the result table is a join among several entities, a join of the corresponding *Description* frames will be established.
- If the result table has one column and several rows, the *List* frame is chosen.
- If the result table has multiple rows and columns, the *Statistical* frame or the *Evaluation* frame is the most appropriate.

If the user's question and the shape of the result table do not provide enough information for selecting frames, values in the result table will be used to fill in slots of all candidate frames. The frames whose required slots[3] cannot be filled will be eliminated.

4.2 A Slot Filler

After frames have been selected, the Slot Filler has to generate text using the rule set mentioned in Section 3. This is a top-down generating algorithm.

Our target is to create a variety of output texts, but to keep the algorithm complexity low. Therefore, when selecting rules from the rule set, if two or more rules are satisfied, the rules will be chosen by the following policy:

- All rules whose the LHS starts with *[frame_* are chosen. This policy is used

[2] The SQL query is generated by the Query Translator.

[3] A required slot is the slot that must be filled in by text or values. Otherwise, the frame that contains this slot cannot be used.

to guarantee the flexibility of the output texts.

- If the LHS of the rule does not start with *[frame_*, the system will randomly choose one rule among the satisfied rules that have not been used in expanding the chosen frame. If all satisfied rules have been used, they are reselected another round by the same method. This strategy is used to prevent the combination explosion and to make sure that a rule is not repeatedly used all the time.

The Slot Filler needs a functional word dictionary[4] to map values in the result table with frame's slots. For example, the table column *Full name* is mapped with the attribute *studname* of the entity *tbl_student*. Therefore, values in this columns are filled in slots *[studname]* of the considering frames.

Several frame types (e.g., the *List* frame) reuse some parts of the user's question in their content. Therefore, a syntactic parser is integrated with the TGEN to get the syntactic structure of sentences.

The slots that do not have values to fill in will be removed from the frame. This action may cause sentential fragments (e.g., a sentence without a verb phrase) in the output texts. For that reason, the TGEN needs a Syntactic Refiner to solve this problem. The Syntactic Refiner will be introduced next.

4.3 A Syntactic Refiner

The purpose of the Syntactic Refiner is to produce grammatical sentences from the outputs of the Slot Filler. It first parses the outputs of the Slot Filler to detect ungrammatical sentences. In order to do that, the Syntactic Refiner locates positions of NPs and VPs in the Slot Filler's outputs by tracing the applied generation rules. Then, it checks the syntax of sentences given their NPs and VPs.

If a sentence lacks of a major part such as an NP or a VP, the Syntactic Refiner will combine it with its adjacent sentences. If it is not success, the ungrammatical sentence will be removed from the output texts.

[4] The functional word dictionary is used to store relations between a word/phrase and its role in the database. For example, the word *student* corresponds to the entity *tbl_student* in the student management database.

4.4 An Answer Generator

Although the output texts of the Syntactic Refiner are grammatically correct, it may not be fluent. There are some reasons for this problem: the sentences can be too short or too long; some words are repeated several times; etc. The Answer Generator has to refine the texts so that it can be as natural as possible. We only deal with repeated words in this research. The Answer Generator replaces repeated words by its synonym or reference words. A thesaurus, which stores semantic relations among words, is used in this process.

We will illustrate the working process of the TGEN by examples given in Section 5.

5 Examples of the Generating Process

In this section, we consider two examples corresponding to two typical frame types. One frame type does not need syntactic information from user's questions (e.g., the *Description* frame – Example 1), and another frame type does (e.g., the *List* frame – Example 2).

5.1 Example 1

User's question:
(1) What is the name of the student whose student number is 20050245? Which class and faculty does this student studies?

The result table returned by the SQL query is:

Stud. name	Class	Faculty
Pham Thanh	BK20	Information Technology

The following steps are carried out by the TGEN:

Step 1: Selecting frames

In this example, the system cannot find keywords that signal question types. It then looks at the result table and finds that the result table has several columns and one row. Therefore, the *Description* frame is chosen. Based on the SQL query, TGEN detects that values in the result table are from three attributes *tbl_Student.StudName, tbl_Class.Classname,* and *tbl_Faculty.FacName.* It automatically generates a new frame for the answer by connecting the frames *[frame_Student], [frame_Class],* and *[frame_Faculty]* through entities' keys.

As mentioned in Section 3, only frames' parts that relate to attributes of the result table are considered to be put in the new frame. During the generating process, the entities' keys are often replaced by their corresponding name attributes (e.g, [*StudId*] is replaced by [*studname*]). In order to select the suitable frames' parts, the system has to find the appearance of at least two among three attributes *tbl_Student.StudName, tbl_Class.Classname,* and *tbl_Faculty.FacName* in the above three frames. If TGEN cannot find these parts, it replaces name attributes by the corresponding key attributes and finds the appearance of these attributes (e.g., finding the appearance of *tbl_Student.StudName* and *tbl_Class.ClassId* or *tbl_Student.StudId* and *tbl_Class.Classname*).

The rule set involving *tbl_Student* has two rules whose LHS is *[frame_Student]*. The RHS of these two rules has one part that involves *tbl_Student.StudName* and *tbl_Class.ClassId*: *[studname] [vp_classId]*.

The rule set that describes relations among attributes of the *tbl_Class* is:

(8) [frame_Class] → [np_Classname] [vp_ClassIdC]. [np_Classname] [vp_FacIdC].
(9) [np_Classname] → Class [classname]
(10) [vp_ClassIdC] → has the code [ClassId]
(11) [vp_FacIdC] → belongs to the faculty [FacId]

The rule set that describes relations among attributes of the *tbl_Faculty* is:

(12) [frame_Faculty] → [np_FacName] [vp_FacIdK], [np_Address], [np_Tel].
(13) [frame_Faculty] → [np_FacName] ([np_FacIdK]) [vp_Address]. [s_Tel].
(14) [np_FacName] → faculty [facname]
(15) [vp_FacIdK] → has the code [FacId]
(16) [np_Address] → address [Address]
(17) [np_Tel] → telephone number [Tel]
(18) [vp_Address] → has the address [Address]
(19) [s_Tel] → Contact number: [Tel]

The rule set of *tbl_Class* has one rule whose LHS is *[frame_Class]*. The RHS of this rule has one part that involves *tbl_Class.Classname* and *tbl_Faculty.FacId*: *[np_Classname] [vp_FacIdC]*. The rule set of *tbl_Faculty* has one rule whose LHS is *[frame_Faculty]*. However, the RHS of this rule does not contain any part that relates at least two among three entities student, class, and faculty. Finally, TGEN has collected the following parts: *[studname] [vp_classId]* and *[np_Classname] [vp_FacIdC]*. The new frame rule is:

(1') [new_frame] → [studname] [vp_classId]. [np_Classname] [vp_FacIdC].

The rules that will be used are collected from the rule set of *[frame_Student]* and *[frame_Class]*.

(6) [vp_classId] → is a student of the class [classId]
(7) [vp_classId] → studies in the class [classId]
(9) [np_Classname] → class [classname]
(11) [vp_FacIdC] → belongs to the faculty [FacId]

Since the result table does not contain information about key attributes of entities, the slot corresponding to the key attributes in the above rules are replaced by the corresponding name attributes.

(2') [vp_classId] → is a student of the class [classname]
(3') [vp_classId] → studies in the class [classname]
(4') [np_Classname] → class [classname]
(5') [vp_FacIdC] → belongs to the faculty [facname]

Step 2: Filling in frames' slots

TGEN generates text by expanding the RHS of the rule (1'). Since the columns *Stud. name, Class, Faculty* are from three attributes *tbl_Student.Name, tbl_Class.Name,* and *tbl_Faculty.Name,* values in the first, second, and third column are filled in the slots *[studname], [classname]* and *[facname]*. Since there are two rules (2' and 3') that can be applied to *[vp_classId]*, TGEN will randomly chooses one of them. The result generated after this step is:

(*1a) Pham Thanh is a student of the class BK20. The class BK20 belongs to the faculty Information Technology.*

Step 3: Refining the syntax of sentences

After filling in frames' slots, the system checks the syntax of sentences in these frames. Since both sentences are syntactically correct, the system does not modify the output of Step 2.

Step 4: Refining the output texts

The system finds a full name is repeated at the second sentence of the output text. Therefore, the system replaces the repeated name by a reference string, which is *this class* in this case.

After being refined by Step 4, the output now becomes:

(1b) Pham Thanh is a student of the class BK20. This class belongs to the faculty Information Technology.

5.2 Example 2

User's question:

(2) *Who got the mark 10 in the Database subject?*

The syntactic parser integrated in the TGEN determines that the interrogative pronoun *Who* is the noun phrase of the question, and *got the mark 10 in the Database subject* is the verb phrase of the question.
Let us consider the case when the query returns only one value, as shown in the result table:

Full name
Nguyen Thuy Linh

In this case, the system chooses the *Single_value* frame (Step 1). To produce an answer, the interrogative pronoun of the question is replaced by the value *Nguyen Thuy Linh* (Step 2). The output of Step 2 is:

(2a) *Nguyen Thuy Linh got the mark 10 in the Database subject.*

Steps 3 and 4 do not modify the above output. Therefore, Sentence (2a) is the final answer.

Consider the case when the result table has several values, as shown in the table below:

Full name
Nguyen Thuy Linh
Dinh Thu Van

The four steps being carried out by the TGEN are:

Step 1: Selecting frames

The answer applies the *List* frame.

Step 2: Filling in frames' slots

The VP of the answer is *are*. The NP of the answer will be constructed by the following formula:
[NP1 in plural] [relative pronoun] [VP of the question]
in which the *NP1* is generated as follow:
The system determines where values in the columns *Full name* come from by looking at the SQL query. It returns a pair (entity, attribute), which is (*tbl_student*, *name*) in this case. The system then interprets the code *tbl_student* to *student* by searching in the functional word dictionary. *NP1* now is *student*. Since the query returns several values, the word *student* is put in the plural form.
Who is chosen to be the *[relative pronoun]* in this example. Therefore, the output of Step 2 is:

(2b) *Students who got the mark 10 in the Database subject are*

- *Nguyen Thuy Linh*
- *Dinh Thu Van*

Sentence (2b) is the final answer since Steps 3 and 4 do not modify the above output.

6 Implementation Discussion and Experiments

A prototype of the TGEN has been implemented for Vietnamese language. Since text generation is our focus in this research, we assume that the Query Translator has already translated the user's question into a SQL query. A Vietnamese syntactic parser (Le-Thanh et al., 2000) is integrated into the system in order to get the syntactic structure of sentences. The program is written in Java and the database management system is implemented in SQL Server 2000.
The TGEN produces one or several textual answers for a user's question, depending on the number of answer frames a question has. The current version of the system has not evaluated the quality of the answers yet. Instead, all possible answers are displayed in an editable interface so that the user can select, modify, save to file, and print the textual answers.
The syntactic rules and the thesaurus in the TGEN are domain independent, whereas the rule set and the functional word dictionary are strictly related to the student management domain. To increase the portability of the system, all knowledge sources are stored in separate data files. If the TGEN is applied to other domains, we only need to modify the content of the files corresponding to the generation rule set and the functional word dictionary.
The TGEN architecture can be applied to a variety of other languages. When changing to another language, we only need to change the knowledge sources of the system. These knowledge sources include the syntactic rules, the thesaurus, the generation rule set and the functional word dictionary.
Some experiments with our prototype system are shown below.

Question 1:

(3) *Điểm thi cao nhất trong kỳ thi Tin học học kỳ 1 năm 2006-2007 là bao nhiêu? Ai đạt điểm cao nhất?*

What is the highest mark in the Informatics examination in Semester 1 of the academic year 2006-2007? Who got the highest mark?

The answer for Question 1 is:

(3a) *Điểm thi cao nhất trong kỳ thi Tin học học kỳ 1 năm 2006-2007 là 9. Bành Quỳnh Mai đạt điểm cao nhất.*

The highest mark in the Informatics examination in Semester 1 of the academic year 2006-2007 is 9. Bành Quỳnh Mai got the highest mark.

Question 2:
(4) *So sánh tỷ lệ sinh viên giỏi của lớp BK20 và BK21.*

Compare the percentage of excellent students of the classes BK20 and BK21.

The answer for Question 2 is:

(4a) *Tỷ lệ sinh viên giỏi của lớp BK20 là 20%. Với lớp BK21, tỷ lệ này là 25%. Ta có thể thấy tỷ lệ sinh viên giỏi của lớp BK21 cao hơn lớp BK20.*

The percentage of excellent students of the class BK20 is 20%. With the class BK21, this percentage is 25%. We can see that the percentage of excellent students of the class BK21 is higher than the class BK20.

(4b) *Tỷ lệ sinh viên giỏi của lớp BK20 là 20%. Với lớp BK21, tỷ lệ này là 25%, cao hơn so với lớp BK20.*

The percentage of excellent students of the class BK20 is 20%. With the class BK21, this percentage is 25%, higher than the class BK20.

7 Conclusions

We described in this paper the TGEN architecture that allows the implementation of a text generator for generating answers from query result tables of a DBMS. The TGEN is not based on a deep natural language generation approach. Instead, it uses a hybrid one. It combines a set of predefined structures with deeper NLG techniques, including (i) using generation rules to generate text; (ii) checking the syntax of sentences; (iii) replacing repeated words by their synonyms or reference words.

The current prototype of the TGEN can produce flexible and grammatical outputs. It proves that a hybrid approach is feasible for this kind of applications while a deep NLG approach is still far to be reached.

To improve the system performance, future work includes: (i) expanding the rule set to deal with a variety of question types; (ii) researching methods to improve the coherence and the fluency of output texts; and (iii) defining criteria to automatically evaluate the outputs.

Acknowledgement

The author gratefully acknowledges the receipt of a grant from the Flemish Interuniversity Council for University Development Cooperation (VLIR UOS) which enabled the research team to carry out this work.

References

Núria Bertomeu, Hans Uszkoreit, Anette Frank, Hans-Ulrich Krieger and Brigitte Jörg. 2006. *Contextual phenomena and thematic relations in database QA dialogues: results from a Wizard-of-Oz Experiment.* Proceedings of the HLT-NAACL 2006 Workshop on Interactive Question Answering, New York.

Tiziana Catarci, Maria F. Costabile, Stefano Levialdi and Carlo Batini. 1997. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260.

ELF Software Co. 2001. *Access ELF: the amazing software that lets you communicate with Microsoft Access in plain English*. http://www.elfsoft.com/ns/prodserv.htm (Last Updated: Nov., 2001). ELF Software Co. 210 W 101 St. NYC NY 10025

Catalina Hallett, Richard Power and Donia Scott. 2005. *Intuitive Querying of e-Health Data Repositories*. In Proceedings of the 4th UK e-Science All Hands Meeting, Nottingham, UK. S.J. Cox (ed.).

Huong Le_Thanh, Quang Pham_Hong, and Thuy Nguyen_Thanh. 2000. An approach to automatically analyze syntax of Vietnamese text. *Journal of Informatics and Cybernetics*, Volume 15, No.4.

Hui Liu. 1995. *A Visual Interface For Querying a CASE Repository*. In Proceedings of 11th International IEEE Symposium on Visual Languages. Darmstadt, Germany.

Olof Torgersson and Göran Falkman. 2002. *Using Text Generation to Access Clinical Data in a Variety of Contexts*. In Proceedings of MIE2002, vol. 90 of Studies in Health Technology and Informatics, pp. 460-465. IOS Press, 2002.

Shan Wang, Xiaofeng Meng, Shuang Liu. 1999. *Nchiql: A Chinese Natural Language Query System to Databases*. In Proceedings of the International Symposium on Database Applications in Non-Traditional Environments (DANTE'99).