Design Tool Combining Keyword Analyzer and Case-based Parser for Developing Natural Language Database Interfaces

Hideo Shimazu Seigo Arita Yosuke Takashima

C&C Information Technology Research Laboratories
NEC Corporation
4-1-1 Miyazaki, Miyamae-ku Kawasaki, Japan, 216
shimazu%joke.cl.nec.co.jp@uunet.uu.net

ABSTRACT

We have designed and experimentally implemented a tool for developing a natural language systems that can accept extra-grammatical expressions, keyword sequences, and linguistic fragments, as well as ordinary natural language queries. The key to this tool's efficiency is its effective use of a simple keyword analyzer in combination with a conventional case-based parser. The keyword analyzer performs a majority of those queries which are simple data retrievals. Since it uses only keywords in any query, this analyzer is robust with regard to extra-grammatical expressions. Since little labor is required of the application designer in using the keyword analyzer portion of the tool, and since the case-based parser processes only those queries which the keyword analyzer fails to interpret, total labor required of the designer is less than that for a tool which employs a conventional case-based parser alone.

1 Introduction

As the number of commercial on-line databases increases, so does user need for *pragmatic* natural language (NL) interface for communicating with those databases. Case-based parsing is an effective approach to constructing NL interfaces to databases [1] [5] [7] [11]. A standard case-based parser consists basically of a pattern matcher and a case base which stores a large number of *linguistic pattern-concept* pairs. In response to a new input query, the pattern matcher searches the case base for any matching *linguistic patterns*. If one is found, its *concept* portion is output as a semantic representation of the given input query. Though case-based parsing makes it easy to construct domain dependent NL interfaces, it has several serious drawbacks:

- The application designer who uses it must define all possible linguistic patterns.
- The application designer must also define a concept portion to correspond to each defined linguistic pattern.
- Since such pattern-concept definitions will be highly dependent on the nature of the specific application, they must be newly defined for each target system.

In this paper, we propose a novel NL interface model, CAPIT (Cooperative Analyzer and Parser as Interface Tool). It is a self-contained NL interface building tool for relational-like databases, and it integrates NL processing mechanisms with the mechanism used for the incremental acquisition of knowledge needed in that NL processing. CAPIT combines a simple keyword analyzer, KBP(Keyword-Based Parsing module), with a case-based parser, CBP(Case-Based Parsing module). KBP extracts only keywords from an input sentence, and constructs a meaning for the sentence from them. Since NL queries to on-line databases tend to be simple and straightforward, KBP can interpret a majority of those queries. However, because it constructs the meaning only from the keywords, KBP sometimes fails to interpret them. The case-based parser (CBP) is a supplemental module to KBP. CBP is a conventional case-based parser. It consists of a pattern matcher and a case base. Linguistic pattern-concept pairs are stored in the case base. CBP must process *only* those queries which KBP fails to interpret correctly. Since an application designer do not have to define all the possible linguistic patterns, his/her labor required to define linguistic pattern-concept pairs is less than that for a tool which employs a conventional case-based parser alone.

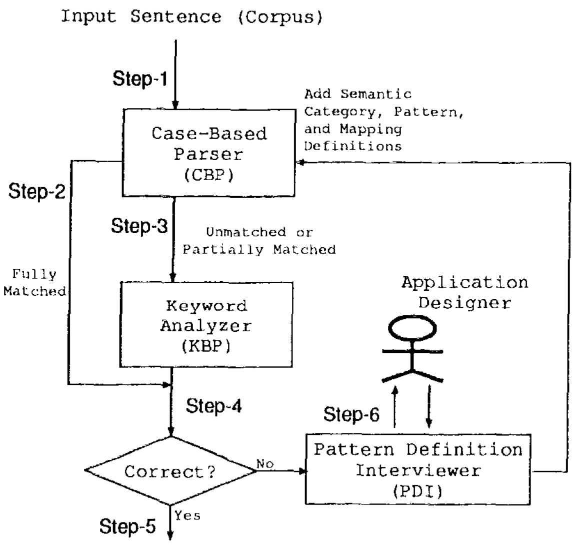


Figure-1: CAPIT Flow

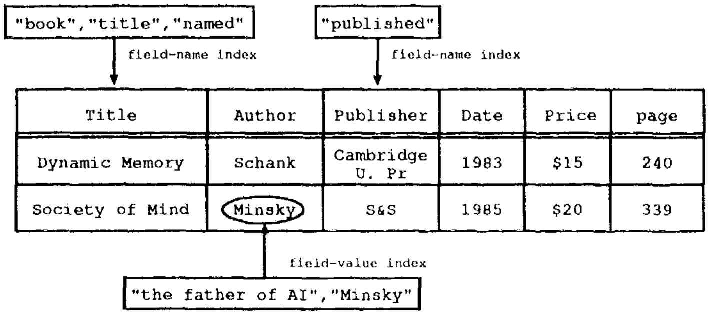
We analyzed KBP's interpretation failures, and categorized the types of KBP's interpretation failures. We regard defining pattern-concept pairs for CBP as repairs of KBP's interpretation failures. We defined four repair types which are corresponding to KBP's typical interpretation failures. When an application designer encounters KBP's interpretation failure, he/she analyzes it, then selects the best and easiest repair type. Such a repair task is accomplished interactively between the application designer and the Pattern Definition Interviewer module (PDI).

2 CAPIT Flow

We have been collecting Japanese corpora which untrained users typed from computer terminals in order to access on-line databases. We found that the large part of the corpora are "Pass me salt" like simple data retrievals from databases. Many sentences have simple grammatical or extra-grammatical structures. Complex linguistic patterns are very rare. One extreme example is just a sequence of keywords like, "Dynamic Memory author", instead of asking "Who is the author of the book titled Dynamic Memory?". We hypothesized that the processing mechanism for such simple expressions is different from a processing mechanism for grammatical expressions. The two parsing module structure of CAPIT reflects this hypothesis.

Figure-1 describes the flow of CAPIT. First, the application designer who develops a NL interface using CAPIT collects the corpora of users' queries in the target domain. A query of the collected corpora is given to CAPIT one by one. The case-based parser (CBP) tries to interpret the sentence (Step-1). If CBP finds a fully matched linguistic pattern in its case base, the corresponding concept is output as the meaning for the input sentence (Step-2). If CBP can not find any matching pattern, the NL query is passed to the keyword-based parsing module (KBP). If CBP finds a pattern which matches with a part of the query in its case base, CBP replaces the matched part of the NL query with the corresponding concept, then passes the modified NL query to KBP (Step-3). KBP extracts only keywords from the query, and constructs its meaning (Step-4). KBP always constructs the meaning for a given sentence.

The meaning generated by CBP and/or KBP, is



Title	Author	Publisher	Date	Price	page
Dynamic Memory	Schank	Cambridge U. Pr	1983	$15	240
Society of Mind	Minsky	S&S	1985	$20	339

Table-1 : A Database Example

shown to the application designer. The application designer judges whether or not the interpretation is correct (Step-5). If it is correct, the examination using this NL query finishes, and the next NL query is taken from the corpora for the next examination. If it is not correct, the Pattern Definition Interviewer module (PDI) is activated. PDI asks the application designer for the correct interpretation of the NL query. He/she defines linguistic patterns and/or semantic concepts and/or the mappings between linguistic patterns and semantic concepts for the NL query (Step-6). The new definition is stored in KBP's knowledge base and/or CBP's case base. Next time CAPIT encounters the same query or similar queries to the query, it succeeds in interpreting the queries correctly.

After numbers of such examinations, CBP's case base becomes rich, and the NL interface application can be released.

3 KBP Mechanism

This section describes the KBP mechanism, using a simple example. Table-1 shows a simple CAPIT target database example. Linguistic patterns are attached as indices which refer to specific fields and the values of specific fields of records in the table. For example, the indices to the "Title" field are "book", "title", "book name", "named", etc. We call an index to a field name ***field-name index***. An index attached to the value of a field of a record is called ***field-value index***. For example, "the father of AI" is a field-value index to "Minsky" which is the value of the "Author" field in a specific record. Values of each field of each record is itself a field-value index. For example, "1983" is a field-value index to the value of "Date" field in a record. Field-name indices and field-value indices are stored in KBP's knowledge base.

KBP always regards the meaning for a given NL query as an imperative, "***Select records in a table which satisfy specific conditions, and return the value of the requested fields from the selected records***". The imperative is represented in SQL:

```
SELECT field-k, field-l, ...
FROM target table
WHERE field-i = value-i,
      field-j = value-j .......;
```

The KBP algorithm to generate the SQL expression from a NL query is as follows:

1. KBP extracts only field-name indices and field-value indices from a given NL query. The rest of the NL query are abandoned.
2. When a field-name index is extracted, its referring field name is kept as a SELECT-clause element.
3. When a field-value index is extracted, its referring field value and the field name of the field value are kept as a WHERE-clause element, in the form of (field name = field value).
4. After all extracted indices are processed, all SELECT-clause elements and WHERE-clause elements are merged. Then, they are assigned into a SELECT-FROM-WHERE structure.

Next, we explain this algorithm, using a NL query example.

S1: "Show me the books published by S&S".

KBP extracts only "book", "published" and "S&S" from S1. "Book" is a field-name index to the "Title" field. "Published" is a field-name index to the "Publisher" field. Since "S&S" is a field-value index to the value of the "Publisher" field, the WHERE-clause element, (Publisher = S&S) is kept. From these indices, the following SQL command is generated:

```
SELECT Title, Publisher
FROM Table-1
WHERE Publisher = S&S;
```

The SQL command is evaluated, and its answer is returned. The answer is "Society of Mind" and "S&S". They are the reply to the above query.

The actual KBP has several heuristic rules to select SELECT-clause elements and WHERE-clause elements. For example, the right answer to S1 is just "Society of Mind". "S&S" must not be produced. With the actual KBP, a heuristic rule suppresses the production of "S&S" in the above example.

Though the actual KBP is more complex than this simple explanation, it is still very simple [2]. Since KBP constructs a query meaning from only keywords in a NL query, it can treat extra-grammatical expressions, keyword sequences and linguistic fragments, in the same way as treating ordinary natural language queries. For example, even the following strange queries on Table-1 are acceptable by KBP; "Publishers?", "Dynamic Memory author", "When the book named Society of Mind appear?", "Society of Mind, how much", etc.

4 The Role of CBP

4.1 The Situations KBP Fails to Interpret

KBP can perform a majority of those queries which are simple data retrievals. So, in what kind of situations does KBP fail to interpret? CBP processes only those queries which KBP fails to interpret. The application designer must define pattern-concept pairs which CBP uses to interpret such queries. Therefore, we have to know the limitations of KBP's interpretation capability. The followings are KBP's typical failure cases.

Failure-1 Cases an application designer forgot to define necessary patterns as indices:
If a necessary linguistic pattern is not defined as either field-name index or field-value index, KBP can not interpret concerning NL queries correctly.

Failure-2 Cases a NL query includes idiomatic expressions or spatial expressions:
KBP can not generate correct meanings, if idiomatic expressions like "greater than 100", or spatial expressions like "the switch between A and B" are included in a NL query.

Failure-3 Cases the meaning for a NL query is not represented in the form of SELECT-FROM-WHERE:
KBP assumes that any NL query is translated into a SELECT-FROM-WHERE structure. If a NL query has a different SQL structure, like SELECT-FROM-GROUP BY-HAVING, KBP can not generate a correct meaning. For example, a NL query like "Select author and its amount which is bigger than 1000" are represented with the SELECT-FROM-GROUP BY-HAVING structure.

Failure-4 Cases the meaning for a NL query can not be represented in SQL language:
If a NL query is a meta-level question for the target database, like "What kind of information can I get from this?", KBP can not interpret it.

Failure-5 Cases KBP generates many candidate interpretations of a NL query:
Since KBP generates the meaning for a NL query using only keywords in the query, it sometimes generates not only a correct meaning but also wrong meanings. For example, KBP generates several different meanings from the following query; "Show me the publisher of the book titled L.A.".

In order to avoid these KBP's failures, when KBP encounters these failures, the application designer must repair the failures, by enriching and modifying either KBP's knowledge base and/or CBP's case base. Such a failure-repair mechanism is analogous to those of case-based reasoning [6] [8].

4.2 Repairs of KBP's Failures

There are four repair types of the KBP's failures. Three of the four are realized by defining a new linguistic pattern-concept pairs in CBP's case base. Failure-5 is solved by either of the four types.

Repair-1 To define a linguistic pattern as either a field-name index or a field-value index:

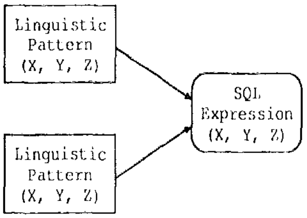


Figure-2: Linguistic Pattern-SQL Pair in CBP for Repair-3

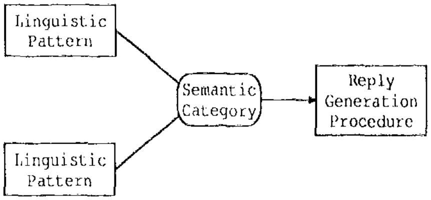


Figure 3: Linguistic Pattern-Semantic Concept Pair in CBP for Repair-4

This is corresponding to Failure-1, and is the easiest of the four repair types.

Repair-2 To define a pattern-concept pair, where the concept part is represented as SELECT-clause elements and/or WHERE-clause elements:
This is corresponding to Failure-2. This is useful to define idiomatic expressions or spatial expressions. Suppose that KBP could not interpret a NL query which included an expression, "price is more than $100, and less than $200". The application designer judges that the part of the query must be defined as a pattern-concept pair. Then, he/she defines a new pattern-concept pair:

[Definition-1]
If a pattern sequence is:
["field-name(Field), [1] {Field is-type-of numerical}, [2] more than, number(N1), less than, number(N2)"],
do the followings:

(1) to keep a field name, "Field", as a SELECT-clause element, and

(2) to keep an expression, " Field > N1, Field < N2", as a WHERE-clause element.

This definition means selecting records whose "Field" has the value more than N1 and less than N2, and returning the value of "Field" of the selected records.

Repair-3 To define a pattern-concept pair, where the concept part is represented as an SQL expression which is not SELECT-FROM-WHERE:
This is corresponding to Failure-3. The application designer must enumeratively define a new SQL structure corresponding to a given linguistic pattern (See Figure-2).

Repair-4 To define a pattern-concept pair, where the concept is represented as a semantic concept which is a meta-level expression for the target database and can not be defined as an SQL form:
This is corresponding to Failure-4. CAPIT provides a frame-like language to define semantic concepts. The application designer defines a new semantic concept using the language. He/she also defines a reply generation procedure. The procedure is called when the corresponding linguistic pattern is matched with an input query (See Figure-3).

Repair-4 is the most difficult of all repair types for an application designer. In Repair-4, he/she must define not only a new semantic concept, but also the definitions of slots in the semantic concept, the procedures which fill the slots, the relations between the new semantic concept with existing other semantic concepts, various constraints among concepts, etc. However, remember that he/she must carry out such complicated tasks to ***all possible*** linguistic patterns in his/her target domain, if he/she uses the case-based parsing approach alone.

5 Dialogue Example between PDI and an Application Designer

PDI (Pattern Definition Interviewer) is CAPIT's interface to an application designer. A dialogue between PDI and an application designer progresses as follows:

1. PDI shows the application designer a NL query which both KBP and CBP have failed to inter-

[1]A term starting with a capital letter is a variable.
[2]An expression surrounded by a pair of brace ({ and }) is a constraint to be satisfied. It is a meta-level description, and is not regarded as a part of pattern sequence.

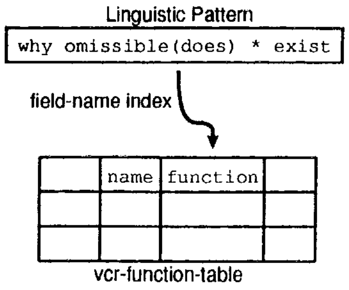


Figure 4: The Repair in the Sample Dialogue

pret. And, it asks him/her to define the correct interpretation to process the input NL query.

2. The application designer analyzes the reason why KBP failed to interpret the NL query.
3. The application designer selects a repair type of the failure, and performs the repair. The definition is stored in either KBP's knowledge base or CBP's case base. Here, he/she can generalize/modify the linguistic pattern, using linguistic pattern generalization/modification operators [10].
4. PDI retries interpreting the NL query again, and asks the application designer whether or not the new interpretation is correct. If it is correct, the definition process of the NL query ends. If it is not correct, go back to 1.

Next, we show a typical sample dialogue between PDI and an application designer. The situation is that the application designer is developing a guidance system which can understand various natural language queries on a specific commercial VCR. The guidance system has an internal database containing data about the functions and the elements of the specific VCR. Each of them is represented its features in a record of the vcr-function-table (Figure-4). The dialogue is an example of Failure-2 and Repair-2. In this example, KBP and CBP are cooperatively generating the meaning for a given sentence.

Suppose, CAPIT is trying to interpret a new input sentence,

S2: "Why does PAUSE exist?"

Since CBP finds no matching pattern, S2 is sent to KBP. KBP extracts keywords from the sentence. Then, KBP generates its meaning. The KBP's interpretation and its generating meaning is shown to the application designer. He/she rejects them. He/she defines a new linguistic pattern which matches with the part of S2,

"why omissible(does) $*$ exist?"

as a field-name index to the "function" field of the target database (See Figure 4). Here, "omissible" is a linguistic pattern modification operator [10], and the special symbol, "$*$", in a linguistic pattern, is a CAPIT's pattern definition notation, which means that it matches with any sequence of words. This definition means that the reason why a specific element exists is described in the "function" field of its corresponding record. After the designer defines the repair of KBP's failure, PDI tries to interpret the same sentence again. This time, since CBP matches "why omissible(does) $*$ exist" with a part of the S2 sentence, CBP replaces the matched part of the S2 sentence with its corresponding concept, that is the "function" field. As a result, the input sentence is transformed into,

S2': "field-name(function) PAUSE ?".

The transformed input sentence is passed to KBP. KBP extracts keywords from the input sentence. The extracted keywords are field-name(function) and field-value(PAUSE). KBP generates a new SQL expression, which is different from the previous one. The application designer judges if the new interpretation is right.

[PDI] Next Sentence is: "Why does PAUSE exist?"
[CBP]: Unmatched!
[KBP]: Extract Keywords:
"PAUSE" is field-value index of "name".
[KBP]: Meaning:
(SELECT $*$ FROM vcr-function-table WHERE name = PAUSE)
[PDI]: ANSWER:
Its NAME is PAUSE. Its TYPE is SWITCH, ...
[PDI]: CORRECT? – > no.

[PDI]: Please define the correct interpretation. – >
define-field-name-index(
[why, omissible(does), $*$, exist],
field-name(function)).

[PDI] Retry Sentence: "Why does PAUSE exist? "
[CBP]: Replaced to:
[field-name(function), PAUSE]
[KBP]: Extract Keywords:
"PAUSE" is field-value index of "name".

[KBP]: Meaning:
(SELECT function FROM vcr-function-table WHERE name = PAUSE)

[PDI]: ANSWER:
Its FUNCTION is ...
[PDI]: CORRECT? – > yes.

6 In Conclusion

The proliferation of commercial on-line databases has increased to demand for natural language interfaces that can be used by untrained people. Real world queries include not only fully grammatical expressions but also such abbreviated expressions as a sequence of keywords, etc [9] [3]. Users will not use a NL interface unless it can also interpret such queries, and CAPIT has that capability

Speed is another important issue. Telephone charge and database access charge are based on time of use, and users require speed. Users will not use a NL interface unless its response time is fast enough. NL interfaces designed with CAPIT are extremely fast. Users' queries are responded within a second.

Ease of development and maintenance is also important. CAPIT is a combination of a keyword analyzer and a case-based parser. Since little labor is required of the application designer in using the keyword analyzer portion of the tool, and since the case-based parser processes only those queries which the keyword analyzer fails to interpret, total labor required of the designer is less than that for a tool which employs a conventional case-based parser alone. With CAPIT, it is possible to design an entirely new NL interface within a matter of weeks.

References

[1] Arens, Y., "CLUSTERS: An Approach to Contextual Language Understanding", Rep. UCB/CSD 86/293, Ph.D. Thesis, 1986.

[2] Arita, S., Shimazu, H., Takashima, Y., "Simple + Robust = Pragmatic: A Natural Language Query Processing Model for Card-type Databases", Proc. of the 13th Annual Conference of the Cognitive Science Society, 1992.

[3] Carbonell, J.G., and Hayes, P.J., "Recovery strategies for parsing extragrammtical language", Technical Report CMU-CS-84-107, Dept. of Computer Science, CMU, 1984.

[4] Cox, C.A., "ALANA Augmentable LANguage Analyzer", Rep. UCB/CSD 86/283, 1986.

[5] Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., and Slocum, J., "Developing a Natural Language Interface to Complex Data", In ACM Trans. on Database Systems, 1978.

[6] Kolodner, J., "Retrieval and organizational strategies in conceptual memory: A computer model", Hillsdale, NJ.: Lawrence Erlbaum Associates, 1984.

[7] Martin, C.E., "Case-based Parsing", In *Inside Case-based Reasoning* edited by R. Schank and C. Riesbeck, Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[8] Riesbeck, C.K., Schank, R.C., "Inside Case-based Reasoning", Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[9] Shneiderman, B., "Designing the User Interface", Addison-Wesley Pub., 1987.

[10] Shimazu, H. and Takashima, Y., "Acquiring Knowledge for Natural Language Interpretation Based On Corpus Analysis", Proc. of IJCAI'91 Natural Language Learning Workshop, 1991.

[11] Wilensky, R. et. al., "UC – A Progress Report", Rep. UCB/CSD 87/303, 1986.