NOVI SAD J. MATH
VOL. 28, NO. 3, 1998, 11-19

DESIGN AND IMPLEMENTATION OF THE QUERY LANGUAGE INTERPRETER FOR BIBLIOGRAPHIC DATA RETRIEVAL

Igor Fišl[1], Zora Konjović[1], Dušan Surla[2]

[1]Faculty of Engineering, Trg D. Obradovića 6, 21000 Novi Sad, Yugoslavia
[2]Faculty of Science, Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia

Abstract. This paper presents the object oriented specification of the query language for bibliographic data retrieval as well as its implementation. Implementation is done as Internet client application using programming language Java.

1. Introduction

The basic functions of the Library information system are: *forming bibliographic data, librarian's reporting and documentation, userćs retrieval, circulation of the bibliographic documentation and defining the processing and retrieval standards* [1].

The query interpretation appears as a subfunction of the user's retrieval and bibliographic data forming process. Creation of the document database and index database upon which retrieval process is running is described and implemented in [3]. Specification and implementation of the client side application supporting bibliographic data forming and retrieval is given in [4]. This client application supports issuing queries for text server and receiving the answers in the network environment. Based on issued query, the server forms new record or retrieves existing records in bibliographic database.

The central part of this paper is concerned with query processing and communication with the server. These functions are implemented as the client application in Java programming language. Implementation of the server supporting indexing and retrieval is based on the Oracle database management system and its ConText Option.

In order to make understanding this text easier for, we shall define some basic specific terms: *document*, *record* and *bibliographic data*. The term *document* denotes a source bibliographic unit (book, journal, etc.). *The record* is set of data describing a particular document. *Bibliographic data* is a set of records.

2. The Query Language

The query language in this system is the one similar to the language used in DIALOG system, as it has world-wide application in the systems for bibliographic data retrieval [1]. This language supports two basic query types, each of them allowing subvariants. The first query type is *select* type with subvariants *select*, *selects* and *selectc*. The second one is *expand* type with subvariants *expand* and *expandd*. Within these statements the logical and proximity operators are allowed.

In the bibliographic database the document is represented by its identification and set of prefixes representing the attributes of the document (e.g., name of the (group of) author(s), document title, publication year, publisher, etc.). Each prefix is characterized by its contents called descriptor. The documents in the database are indexed by the descriptors of all prefixes.

2.1 `Select` type Queries

All subvariants of this query type return as the result identifications of the documents satisfying retrieval conditions (hit documents). The simplest subvariant is `select`. This subvariant returns identifications of all documents satisfying the query condition.
A little bit more complex query type is the `selectc` type. This type, just as a `select` type, returns identifications of hit documents, but it additionally supports range of the hit documents (for example: only the identifications of the first four hit documents will be returned).
The conditions within query can be connected with logical operators. Each part of the query that is connected to other with logical operator is called *term*. The `selects` type query, in addition to the identifications of the documents satisfying the "whole" condition, returns the identifications of the documents satisfying each of the terms within the condition.

2.2 `Expand` Type Queries

The `expand` type query allows only terms to appear as condition. These terms are categorized either as qualified or unqualified. The term is qualified if the prefix of the descriptor that should contain the term is specified. The term is unqualified if the prefix of the descriptor that should contain the term is not

specified. Because the retrieval is carried out in a consistent manner, an unqualified term is qualified by all prefixes of the basic index type.
The `expand` subvariant of this query type allows the condition containing only unqualified term. This statement returns descriptors of the prefixes satisfying the query.
The statement `expandd`, similarly to statement `expand`, returns the descriptors of the prefixes satisfying the query. Only qualified terms are allowed within the condition in this statement.

2.3 The Conditions in the Queries

The queries are constructed of terms connected with logical operators. These operators are `and` (conjunction), `or` (disjunction) and `not` (negation). The qualified terms in queries are qualified depending on the type of the prefix index by which the term is qualified. If the prefix index type is the basic one, the term is qualified in following manner:

```
term / <prefix>
```

If the prefix index type is the additional one, the term is qualified in following way:

```
<prefix> / term
```

The wildcards within term are also allowed. The symbol replacing arbitrary string is *, while the symbol replacing the single character is ?. The following examples illustrate usage of the wildcards.

```
select AU="P*"
```

returns identifications of all the documents containing in the descriptor of its AU prefix any word starting with the character P.

```
selects AU="P*" and "M*"/TI
```

This query returns identifications of the documents containing in the AU prefix descriptor any word starting with the letter P, the documents containing in the TI prefix descriptor any word starting with the letter M and the documents satisfying conjunction of these two queries.

```
expandd "M*"/TI
```

returns the contents of the prefix TI descriptors containing the words starting with letter M.

3. Object Oriented Specification

This paper describes object oriented specification of the client side application supporting the following global functions:

- receiving the query from the user;
- checking syntax and semantics of the query;
- translating the query into the text server understandable form;
- sending the translated query to the server through the network;
- accepting and displaying the query results from the server.

The specification is done by using UML (Unified Modeling Language) [5,6]. The class diagram is shown in Fig. 1.

The "main" class (the class containing the method `main`) is the class `Parser`. This class has only two methods: `getString` and `main`. The method `getString` takes the string input from the keyboard. The method `main` controls the processing of the string, returned as the result from the method `getString`.
The class `list` represents generic single connected list. All operations with this list are executed over objects of the class type `ListElem`. To operate with objects of some other class, it is only necessary to inherit the class `list` by the other class. The class `list` has the following methods:

- inserting the element at the head of the list (`addFirst`);
- inserting the element at the tail of the list (`addLast`);
- inserting the element at the specified position of the list (`insertAt`);
- accessing and extracting the first element of the list (`getFirst`);
- accessing and extracting the last element of the list (`getLast`);
- accessing the first element of the list (`peekFirst`);
- accessing the last element of the list (`peekLast`);
- accessing and extracting the element at the specified position of the list (`getElemAtPos`);
- accessing the element at the specified position of the list (`elemAtPos`);

- returning the current element of the list (the element last accessed but not extracted) (`getCurr`);
- returning the next element of the list (the element immediately following the current element) (`getNext`);

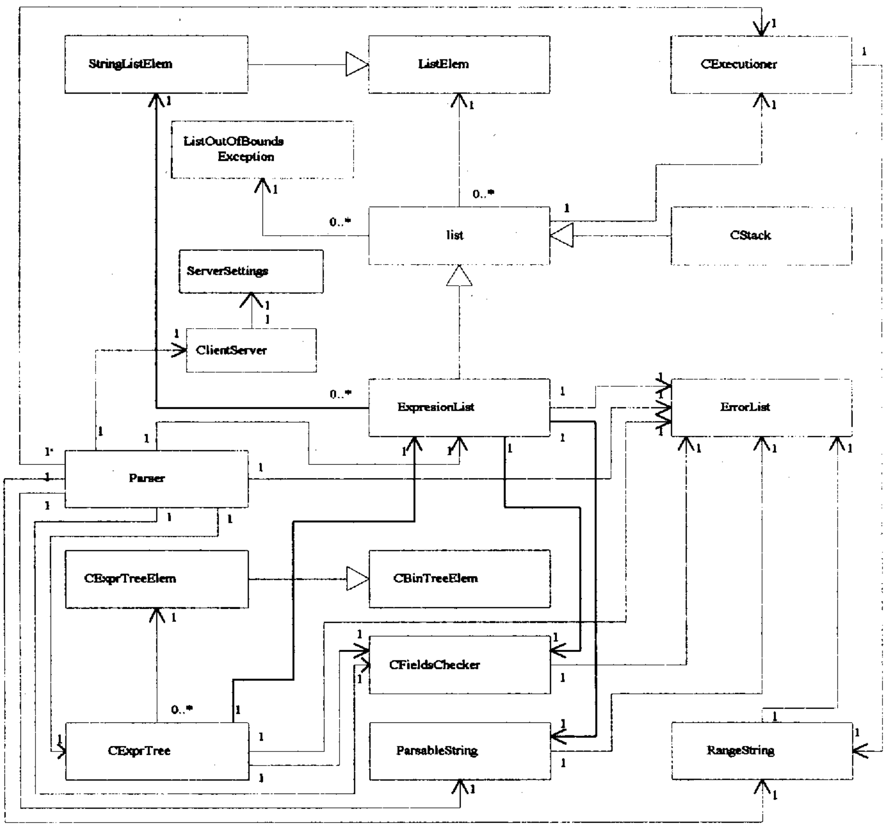

Figure 1. The class diagram of the query language for retrival bibliographic database

- concatenation of the list at the end of the current list (`concat`);
- positioning the current element at the desired position in the list (`seek`);
- getting the empty status of the list (`isEmpty`);

- emptying the list (emptyIt).

The class `ListOutOfBoundsException` is the class whose objects are generated when the exception occurs by inserting the elements at the illegal position of the list (object from class `list`).

The class `Cstack` inherits the class `list`. This class has the methods for inserting an element in the stack (the element is always inserted at the top of the stack) and removing elements from the stack (the element is always removed from the top of the stack).

The class `ListElem` represents the element of the list defined by class `list`. To construct the list of the elements of any other class type it is only necessary to inherit the class `ListElem` by these other classes. According to this specification, the class `StringListElem` inherits the class `ListElem`. The class `StringListElem`, in addition to the fields contained in the class `ListElem` contains the field of type `String` which represents the useful contents of this class type objects. The list whose elements are of class type `StringListElem` is actually the list of strings.

The class `RangeString` for some string checks if the string matches particular format representing input range format. This class is used by commands of type `selestc`, since this type of query does not always return the identifications of all hit documents.

The class `CFieldsChecker` is the class dealing with the prefix names. This class has the following methods:

- the constructor reads information about prefix index type (i.e., which of them (prefixes) has basic or additional, or both index type) from the specified file. Then, these prefixes are stored in the internal data structures of the class to be available for the subsequent work;
- the method checking for the particular index if its prefix type is the basic one (`isBase`);
- the method checking for the particular index if its prefix type is the additional one (`isAdditional`);
- the method returning the name of the next prefix with the basic type index (`getBase`);

- the method returning the name of the next prefix with the additional type index (`getAdditional`);

The class `ErrorList` is the class which records the errors occurred during the execution of the program. The input data format errors are stored in this class. This class writes all error messages into a specified file and records if the error occurred.

The class `ParsableString` is the class used for parsing the strings containing tokens separated by specified delimiters and in which the tokens appear in the sequence created following defined rules. This class supports splitting the string into tokens only if the delimiters separating the tokens are given. When calling the constructor of the class `ParsableString` the object from the class `ErrorList` must be assigned to each object from the class `ParsableString`; the object from the class `ErrorList` will be used by the object from the class `ParsableString` to store the errors in the input data detected during parsing phase.

The class `ExpresionList` is derived from the class `list` which means that this class represents the list, in this particular case the list of elements of type `StringListElem`. This class has the method `toPostfix` which creates the list containing the expressions represented by reverse Polish notation (RPN expression) from the list containing the same expressions represented by infix Polish notation (IPN expression). All operations are performed with respect to predefined operators that are used in the expressions. The constructor of this class also requests the assigned object from the class `ErrorList` because of reporting the errors detected during expression translation.

The class `CBinTreeElem` represents the element of the binary tree. To create the binary tree of arbitrary elements it is necessary to define the class inheriting the class `CBinTreeElem` and then use this class as the element of the binary tree. According to this possibility the class `CExprTreeElem` inheriting the class `CBinTreeElem` represents the element of the particular tree which will be used in the program.

The class `CExprTree` represents the binary tree which is needed to solve this particular problem. The object from the class `ErrorList` representing the list recording the error messages is assigned to the class `CExprTree` in the constructor.

The class `Cexecutioner` is the class executing the query generated by the tree. This class establishes the network connection with the server, sends the query requests to the server and returns the results.

The class `ServerSettings` accepts the parameters for the server connection setup. This class has the methods that return the server name and the number of the port assigned to the socket open to the server.

The class `ClientServer` is used to send the query request to the server and receive the results from the server. The method `connect` of this class opens the connection with the server. The method `reset` resets the connection with the server, while the method `close` permanently disconnects the server. The methods `PlainMessage`, `SQLMessage`, `justSend` and `nativeMessage` are used for sending different types of messages to the server. If the retrieval result returned by the server is more than one line length, then the results are accepted using the methods `getRow`, `buffrizeRowInd` and `bufferizeNextRow`. The method `getHits` returns the number of hits, i.e., the number of documents satisfying the query. The method `analyseResponse` analyzes the responses obtained from the server and updates the internal structures of this class.

4. The Implementation

The implementation is carried out using the programming language Java and in full conformance with the given specification. The inheritance from the UML is implemented by standard inheritance mechanism of the Java programming language. The only additional concept of the UML used in this implementation is the navigation. This concept is implemented in the application in two different ways. The first one is that the navigating object has the navigated object as one of its attributes. The second way is that the navigated object is treated as the argument of the function call of the navigating object.
The implementation of the UML inheritance concept in Java language in this application is illustrated by the classes `list` and `ExpresionList` (the class `ExpresionList` inherits the class `list`). The navigation is illustrated by the usage of the class `ErrorList`, which is the parameter of (almost) all constructors of other classes. It is necessary to note here that the class `ParsableString` is realized by using the class belonging to one of the standard packages of the Java programming language, the class `StringTokenizer`.

5. Conclusion

The client side application for bibliographic data retrieval is developed along with the server application. The server application is implemented using Oracle DBMS and its ConText Option.

The client application is developed using the programming language Java. This language is the new one, but still very widely used, particularly on the Internet. The ability of the Web browser to interpret Java scripts and execute Java programs (applets) makes the final goal of this application to be available in the form of the applet. This form of application provides one the possibility to access and retrieve the library database residing anywhere on the Internet.

References

[1] Lazarević, Branislav (Editor). *Forming and Retrieval of the Databases in the System of Scientific and Technological Information of Republic of Serbia*, Minstry for Science and Technology of Republic of Serbia, Belgrade, 1996. (in Serbian)

[2] *Java Language Specification*, Sunm Microsystems Computer Corporation, USA, 1996.

[3] Milosavljević, Branko. *Implementation of the Text Server for Indexing and Retrieval of Bibliographic Data in Oracle Environment*, B.Sc., Faculty of Engineering, Novi Sad, 1997. (in Serbian)

[4] Fišl, Igor. *Design and Implementation of the Client Application for Retrieval of Bibliographic Data in Java Environment*, B.Sc., Faculty of Engineering, Novi Sad, 1997. (in Serbian)

[5] Tričković, Ivana. *Application of the Petri Nets to Specification of the Dynamics of Information Systems*, M.Sc. thesis, Faculty of Science, Novi Sad, 1997. (in Serbian)

[6] *UML Notation Guide, version 1.0,* RATIONAL SOFTWARE Corporation, Santa Clara, CA, http://www.rational.com