



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 17/30

A2

(11) International Publication Number: WO 96/18159

(43) International Publication Date: 13 June 1996 (13.06.96)

(21) International Application Number: PCT/US95/16496

(22) International Filing Date: 7 December 1995 (07.12.95)

(30) Priority Data:
08/350,967 7 December 1994 (07.12.94) US

(71) Applicant: ARCHITEXT SOFTWARE, INC. [US/US]; 10280 Lockwood Drive, Cupertino, CA 95014 (US).

(72) Inventor: SPENCER, Graham; 10280 Lockwood Drive, Cupertino, CA 95014 (US).

(74) Agents: SMITH, Albert, C. et al.; Fenwick & West, Suite 600, Two Palo Alto Square, Palo Alto, CA 94306 (US).

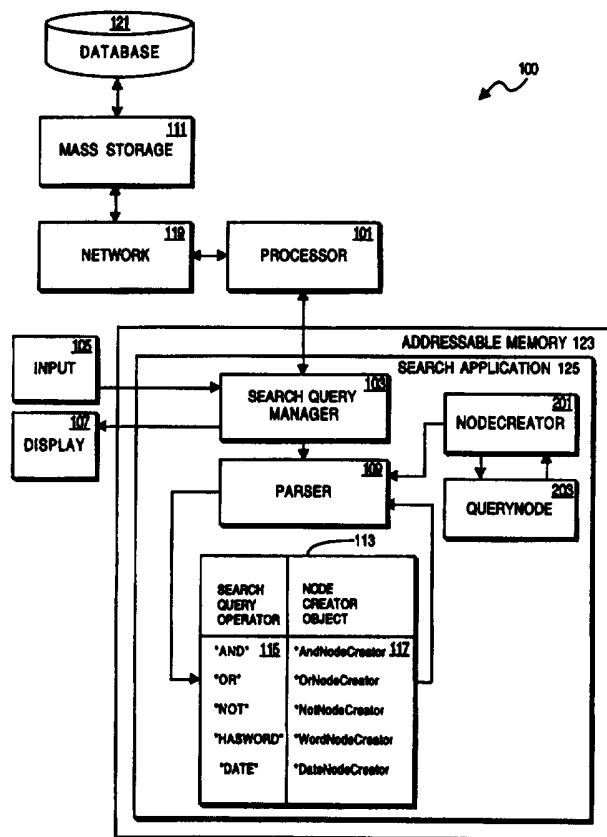
(81) Designated States: AL, AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TT, UA, UG, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, LS, MW, SD, SZ, UG).

Published*Without international search report and to be republished upon receipt of that report.*

(54) Title: IMPLEMENTATION INDEPENDENT EXTENSIBLE QUERY ARCHITECTURE FOR INFORMATION RETRIEVAL SYSTEMS

(57) Abstract

An information retrieval system incorporates an extensible query architecture allowing an applications programmer to integrate new query models into the system as desired. The query architecture is based on an abstract base class of query nodes, or code objects that retrieve records from the database. Specific subclasses for particular query models are derived from the base class. Each query node class includes a search function that iteratively searches the database for matching records. Query node objects are instantiated by associated node creator class objects. A parser is used to parse a search query into its components, including nested search queries used to combine various query models. The parser determines the particular search operator keywords, and the node creator object for instantiating the appropriate query node object for each search operator. The node creator objects return pointers to the created query nodes, allowing the parser to assemble complex hierarchical query nodes that combine multiple query models.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgyzstan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LU	Luxembourg	TD	Chad
CS	Czechoslovakia	LV	Latvia	TG	Togo
CZ	Czech Republic	MC	Monaco	TJ	Tajikistan
DE	Germany	MD	Republic of Moldova	TT	Trinidad and Tobago
DK	Denmark	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	US	United States of America
FI	Finland	MN	Mongolia	UZ	Uzbekistan
FR	France			VN	Viet Nam
GA	Gabon				

IMPLEMENTATION INDEPENDENT EXTENSIBLE QUERY ARCHITECTURE FOR INFORMATION RETRIEVAL SYSTEMS

BACKGROUND

Field of the Invention

This invention relates generally to the field of information retrieval systems and methods, and more particularly, to methods for combining multiple queries through an abstract programming interface.

Background of the Invention

Information retrieval systems typically include a database of records, a processor for executing searches on the records, and specifically adapted application software, such as a database management system, for accepting search queries, managing the processor, and handling the search results. In general, the database can include information such as text documents, financial records, medical files, personnel records, technical documentation, graphical data, or various combinations of such items. In order to effectively search and retrieve desired items, the search application typically supports a limited number of query models, or search operations, specifically designed to operate on the underlying data types in the database. For example, a typical document database, such as a database of news publications, may be organized with each news article as a record, with fields for publication date, author, title, industry category, and body text. A simple search application may then support full text searching for all text fields, individual field searching, such as searching by the date or author fields, and various boolean search operations, such as conjunction, disjunction, and negation. A more sophisticated search application may also support proximity based searching, allowing a user to locate word tuples having specified proximities. This would allow a user, for example, to locate in such a news database, all articles having the words "Clinton" within 25 words of "foreign policy." Thus, proximity searching is specific form of boolean conjunction.

One of the limitation of existing information retrieval systems is the difficulty in combining different query models in a single search. For example, a simple system may allow a user to perform either full text searching, or field based search, but not both in a single query. A user interested in retrieving documents having certain keywords would be unable to simultaneously constrain the documents to those of given date, author, or the like. More robust systems may offer only a limited mixing

1 of field based searching and full text searching, but do not support full integration of
2 non-field based queries. This limitation stems primarily from the query architecture
3 used by the software vendor, with the various different query models having
4 incompatible operations or algorithms.

5 The difficulty in combining multiple query models generally results from the
6 non-extensible query architecture that underlies the information retrieval system. In
7 conventional information retrieval systems, the various query models that are
8 supported, such as text based searching, boolean searching, and the like, are normally
9 implemented with implementation specific code, designed for the specific data types
10 and operations available in the information retrieval system. The software vendor
11 does not provide any capability for an extensible architecture for the search
12 operations or data types available in the system. This is generally because of the
13 implementation specific storage and performance constraints that the vendor has
14 designed into the information retrieval system. Thus, an applications programmer
15 utilizing the information retrieval system will typically be constrained to using search
16 operations that the vendor has provided, and will be unable to add new search
17 operations for new query models or data types. Continuing the prior example of the
18 database of news publications where the software vendor has provided only full text,
19 field, and boolean query operations, an applications programmer would typically be
20 unable to add search operations that retrieved documents based on statistical
21 information, such as the number of references to a given word or set of words, or the
22 frequency of specific references in arbitrary subsets of the database.

23 Another limitation of existing information retrieval systems is that the search
24 algorithms are designed to execute over a substantial portion of the database,
25 returning their results in memory intensive arrays or similar structures. Typically, the
26 software vendor defines the computational design of each search operation,
27 preventing the applications developer from designing more efficient algorithms for
28 implementing a given search operation. This results in limitations on the performance
29 an information retrieval system can deliver.

30 Accordingly, it is desirable to provide a query architecture for an information
31 retrieval system that is extensible and allows for the efficient integration of new
32 query models designed for an open variety of data types and formats. This would
33 allow the information retrieval system to support any arbitrary combination of query
34 models, and further allow the applications programmer to add new query models and
35 data types to the system as needed or desirable. A desirable query architecture
36 should place minimal constraints on the necessary storage and performance needed
37 to perform search operations. This would allow the applications programmer to

1 design the information retrieval system to perform efficiently on a variety of operating
2 platforms.

6 SUMMARY OF THE INVENTION

7 The present invention overcomes the limitations of the prior art by providing
8 an extensible query architecture that allows the creation and integration of new
9 query models into an information retrieval system, and further provides for the
10 efficient combination of multiple query models during individual search operations.
11 The invention comprises a information retrieval system including a search application
12 that has a variety of code module classes, each one for implementing a specific type
13 of query model on particular data types in an attached database. The code module
14 classes, or QueryNodes, all derive from a common QueryNode class which defines the
15 architecture of all the query nodes. The QueryNodes all share a common object
16 interface that allows the system to combine QueryNodes representing different query
17 models into a single search query.

18 Each query node object is capable of performing a search query and
19 describing its results by associating a document score with each document in the
20 database subject to the query. Specifically, each query node object, regardless of the
21 query model that it implements, uses a common signature or interface for accepting
22 input arguments. The interface is designed to allow other query node objects to be
23 part of the argument list. This allows different query models to be combined in a
24 single search query.

25 In order to allow the query nodes to have a common interface, the query
26 nodes include a search function operates consistently for different query nodes. The
27 search function is designed to take as its input a document number representing a
28 document in the database. The query node search function locates a document in the
29 database with a document number greater than or equal to the input document
30 number (thereby ensuring that each document is evaluated only once for a given
31 query model) that has a non-zero document score. The document score is determined
32 by an independent scoring function that can be designed separately from the search
33 function, and which is called by the search function as needed. Separating the
34 scoring function from the search function increases the flexibility and generality of
35 the search function. If a document with a non-zero score is found, the query node
36 updates a document score parameter that is also passed to the query node. Since the
37 search function operatively iteratively, the entire database need not be searched at
38 once, but rather, the execution of the search can be scheduled as necessary, with

1 portions of the database searched in one batch, and other portions searched
2 separately, or not at all.

3 The invention supports the combination of multiple query models by allowing
4 a query node to invoke other query nodes during execution, and to act on the results
5 provide by the invoked query node. The information retrieval system of the
6 invention uses a parser that identifies in an input search query the various operators
7 and arguments that comprise the search query, including multiply nested search
8 operators. For each search operator that the parser identifies, there is a stored
9 association with a code module that creates a query node for executing the search
10 operation represented by the search operator. Thus, for example, for a field based
11 search, such as searching by the date of a record, a search query may take the form
12 "DATE(>07/11/1987)." The parser identifies the search operator "DATE" and then
13 calls a code module called a NodeCreator, that instantiates a code object from the
14 particular query node class for searching date type fields in the database, the query
15 node object being dynamically created for performing the requested search
16 operation. The NodeCreator returns to the parser a pointer to the newly created
17 query node object. Combinations of different types of query models are possible
18 because even though the underlying query models may be different, all query node
19 objects interface with each other and parser in a standard manner. In particular, any
20 query node object can be one of the arguments to another query node object,
21 thereby allowing any combination of child query types to be created by the user.
22 The parser manages the combination of child query node objects to create a single
23 parent query node for execution by the system.

24 The query architecture is extensible because new query models can be
25 implemented by creating new query object subclasses from the QueryNode base
26 class, and providing them with a complementary scoring function. For each new
27 query node class, a new search operator is also created and stored in association with
28 a NodeCreator class object for instantiating an object from the new class whenever
29 the search operator is used.

30 BRIEF DESCRIPTION OF THE DRAWINGS

31
32 Figure 1 is a block diagram of one embodiment of the information retrieval
33 system of the present invention.

34 Figure 2 is an object hierarchy diagram showing the object class hierarchies for
35 the NodeCreator and QueryNode classes.

36 Figure 3 is a flowchart of the method of operating the information retrieval
37 system of the present invention.

1 Figure 4 is a illustration of a hierarchical top node resulting from parsing an
2 input search query.

3 Figure 5 is a flowchart of the method creating a new QueryNode class for use
4 with the information retrieval system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring now to Figure 1, there is shown one embodiment of the present invention for combining multiple queries using an extensible query architecture. The information retrieval system **100** includes a processor **101** operatively coupled to a display **107**, an input device **105**, a network connection **119**, and an addressable memory **123**. In the preferred embodiment the system **100** can be implemented on a Sun Microsystems Sparc workstation, or any other comparable computer, using software in the addressable memory **123** embodying the query architecture of the present invention. The network connection **119** provides access to a remotely situated mass storage device **111** which stores any type of database **121**, such as text documents, financial records, medical records, technical manuals, and the like. The display **107** is of conventional design and provides output for a graphical user interface to a search application **125** stored in the addressable memory **123**. An input device **105**, such as a keyboard, mouse, and the like, is provided for inputting commands and search queries to the search application **125**. The input device **105** may include a voice recognition unit to directly accept spoken search queries or commands.

The addressable memory **123** contains a search application **125** incorporating the query architecture of the present invention. The search application **125** is preferably implemented in an object oriented programming language, such as C++ (which will be used for examples herein), but may be implemented in other programming languages. The search application **125** includes a search query manager **103**, a parser **109**, a search operator association table **113**, a number of NodeCreator objects **201**, and a number of QueryNode objects **203**.

In order to provide an interface to the user for using the information retrieval system **100**, the search query manager **103** is coupled to the input device **105** and the display **107**, and accepts a search query input by a user with the input device **105**. The input search query is eventually processed by the processor **101** to apply it to the databases **121** stored in the mass storage devices **111**, either locally, or on the network **119**. Any results for the search query are returned to the search query manager **103** for outputting on the display **107**.

The search query manager **103** accepts search queries having specific keywords that act as search operators, each search operator associated in the addressable memory **123** with a particular query node class that implements a given query model, as further described below. The search operators may include keywords for creating boolean queries, such as "AND," "OR," "NOT," keywords such as "DATE" and "TIME" for retrieving documents having particular creation or modification dates or times, and keywords such as "HASWORD" (or conventionally

1 the absence of any keyword) for retrieving documents having particular words.
2 Because the query models are extensible, new search operator keywords can be
3 created by the applications programmer to access new query models.

4 Each search query is composed of at least one search operator and zero or
5 more data elements which are the arguments for the search operator. A simple search
6 query is:

7 (HASWORD computer)

8 This search query is a full text search query and would be used to retrieve all
9 documents in the database that have the word "computer" in their text.

10 It is noted that the sample search queries are illustrated here by explicitly
11 including the search operator merely to increase the clarity of the disclosure. In actual
12 practice search operators can be implicitly represented so that the user need only
13 enter the data elements for the search query, with the parser 109 determining the
14 search operators associated with each data element based on predefined parsing
15 rules, field values, and the like. Thus, in one embodiment, the above search query may
16 be enter by the user as merely "computer." Also, the examples here as shown use
17 prefix notation merely to more accurately capture the hierarchical nature of complex
18 search queries; other syntax notations, including infix and postfix may also be used.

19 Where the database includes specific fields, such as a date field, another simple
20 search query could be:

21 (DATE > 03/03/94)

22 This search query operates on a date type field, and returns documents with a
23 date after March 3, 1994. The data elements for a search operator may include other
24 search queries, thereby allowing simple search queries to be nested to form complex
25 search queries, such as:

26 (AND (HASWORD computer) (NOT (HASWORD Intel)) (DATE > 03/03/94))

27 This sample search query is a multiway boolean *and* query, each conjunct itself
28 a search query including a search operator and a data element. This search query is
29 for retrieving documents that have the word "computer" but do not have the word
30 "Intel" in their text, and which have a creation date after March 3, 1994. In this
31 complex search query, the AND is the "parent" query, and the conjuncts are "child"
32 queries. The ability to form complex search queries is useful for efficiently combining
33 query models.

34 The search query manager 103 passes an input search query to the parser 109.
35 The parser 109 is a text parser that decomposes an input search query into its
36 constituent parts, identifying any nested search queries, the search operator in each
37 search query, whether explicitly or implicitly represented, and the data element

1 associated with each search operator, if any. Thus, with the previous complex search
2 query, the parser 109 would identify the following child queries:

3 (DATE > 03/03/94)

4 (HASWORD Intel)

5 (NOT (HASWORD Intel))

6 (HASWORD computer)

7 and the parent query

8 (AND (HASWORD computer) (NOT (HASWORD Intel)) (DATE > 03/03/94))

9 In the preferred embodiment the parser 109 decomposes the search query
10 recursively, though non-recursive parsing may also be performed, to identify the inner
11 most child components, specifically those search queries whose data elements are
12 string values, including numerical values, and not further child queries.

13 The parser 109 has access to a search operator association table 113 that
14 associates the search operator keywords 115 with pointers 117 to NodeCreator 201
15 objects. The table 113 may be implemented as a hash table or other suitable data
16 structure. The NodeCreator 201 objects derive from a NodeCreator 201 base class,
17 each NodeCreator 201 object implementing a method that creates a particular
18 QueryNode 203 object for executing a query of the type specified by the search
19 operator.

20 Figure 2 is an object class diagram for the NodeCreator 201 and QueryNode
21 203 classes. The QueryNode 203 class is an abstract class from which specific
22 QueryNode subclasses can be derived, each QueryNode subclass being used to
23 instantiate a code object that performs a specific search query and returns a value
24 describing its results.

25 One of the benefits of the extensible query architecture of the present
26 invention is that it supports any variety of query models, including full text searching,
27 field based searching, knowledge based searching, statistical searching, and the like.
28 Because the QueryNode 203 class is an abstract class, it provides the basis of the
29 extensible query architecture by allowing the applications programmer to implement
30 new query models specifically designed to meet the search needs of the user, or the
31 database environment by deriving new NodeCreator 201.x and QueryNode 203.x
32 classes from the respective base classes.

33 The QueryNode 203 class has the various members for effecting a search
34 query. Specifically, the QueryNode 203 class has a data element member that is used
35 to store the argument on which the QueryNode 203 object is to perform its member
36 functions.

37 The QueryNode 203 class further has a member function performs the search
38 operation on the database. The search function associates a particular score with

1 each document or record in the database. The score of each document is then
2 evaluated using defined weighting and evaluation functions in order to segregate
3 and rank the documents as to the relevance to the input search query. In the
4 preferred embodiment, this search function of the QueryNode 203 class performs an
5 iterative search by locating a single document with a non-zero score. This function
6 can be illustrated as follows by the following C++ function prototype:

```
7     class QueryNode {  
8         public:  
9             virtual int next(int* doc_num, float* doc_score) {  
10                 code to implement actual query model  
11             }  
12     };
```

13 The next function takes as its primary input a document number `doc_num`.
14 The document number is a unique serialized integer value for each document in the
15 database. The `doc_score` is an input/output parameter, and is an evaluated value
16 for a document with respect to the search query being executed by the particular
17 QueryNode 201 object. In the preferred embodiment the document score is not
18 determined directly by the next function, but is computed by a specific scoring
19 function that is called by the next function. When passed a document number, the
20 next function locates in the database another document whose document number is
21 greater than or equal to the input document number, and whose document score is
22 greater than zero. If such a document exists, then `doc_score` is set to the
23 document score of the located document, and next returns 1 indicating a successful
24 search. If no such document exists, next returns 0. This allows the next function
25 to be evaluated only once for each non-zero document in the database. This results in
26 increased performance for the search operation because the number of non-zero
27 documents resulting from a given search query is typically significantly less than the
28 total number of documents in the database, thereby reducing the search space
29 operated over by the next function.

30 The iterative operation of the next function provides increased flexibility and
31 efficiency over existing search query interfaces. Conventional search routines search
32 many, if not all, documents in a database, either directly, or indirectly through indexed
33 tables or the like, and return an array of document scores. This conventional
34 approach is both memory intensive and algorithmically inefficient: it is memory
35 intensive because 201 it returns an array of scores, many of which will be 0 since the
36 documents are not "hits." Second, returning an array of (`doc_num`, `doc_score`)
37 tuples is algorithmically inefficient because it requires that the search query be

1 evaluated completely over the entire database, thereby consuming a significant
2 amount of processing time. Such as array implementation discourages the
3 applications programmer from implementing other data structures that are better
4 adapted to store search results, due to the overhead involved in converting the array
5 to a desired form. Other conventional scoring interfaces that provide a score for each
6 document in the database, such as `virtual float score(int doc_num)`
7 are also inefficient because the function must be executed once for every document
8 in the database, even though most documents will have a score of 0. Because the
9 `next` function does not attempt to return all matching documents at one time, the
10 applications programmer can implement the search application 125 to effect the
11 `next` function in the most efficient manner, for example, by delaying performing
12 parts of a search query until the system load decreases, or some higher priority
13 process is completed. The iterative nature of the `next` function obviously applies to
14 all child `QueryNodes` in a search query, thus the applications programmer can control
15 the execution of a complex search query, rather than relying entirely on the internal
16 search engine to manage the execution process. Since the applications programmer
17 can assume that most search queries will be executed sequentially, the search
18 application 125 can be optimized for this search pattern while still allowing flexibility
19 for alternative search implementations. In this manner, the query node architecture in
20 implementation independent.

21 Referring again to Figure 2 then, the `QueryNode` base 203 class is used to
22 derive a number of particular `QueryNode` subclasses, each one supporting a particular
23 query model. The basic `QueryNode` subclasses include various boolean query
24 models, such as an `AndQueryNode` 203.1 class, an `OrQueryNode` 203.2 class, and a
25 `NotQueryNode` 203.3 class, a full text query model, such as `WordQueryNode` 203.4
26 class. Other application specific query models can also be derived as needed from the
27 `QueryNode` base 203 class, depending on the type of database being searched. For
28 databases that incorporate various fields, such as date, time, author, document
29 category, and the like, specific query models for each field type can be implemented
30 with appropriate `QueryNode` subclasses, such as `CategoryQueryNode` 203.4 class or
31 `DateQueryNode` 203.5 class. A derived `QueryNode` 203.x subclass may also include
32 additional or data members, such as an array of documents and scores in order to
33 execute a particular query model on the array.

34 Each newly created `QueryNode` 203.x subclass will have the attributes of the
35 `QueryNode` 203 base class. This feature makes the query node architecture
36 extensible. In contrast, in conventional search applications, such as database
37 management systems, word processors, records management, and the like, the number
38 and type of query models that can be used is fixed: the applications programmer can

1 only use the specific query models that the software manufacturer has provided, and
2 cannot create new query models for use in the application.

3 As noted above, the next function of the QueryNode 203 class provides
4 improved performance by evaluating only non-zero documents. This features
5 provides especially improved performance for boolean AND operations, typically the
6 most common complex search operation, because the function is evaluation only
7 once for the number of non-zero documents resulting from the child search query
8 that returns the fewest non-zero documents. For example, the next function of a
9 multiway AndQueryNode with an arbitrary number of children QueryNodes may be
10 implemented as follows:

```
11 class MultiAndQueryNode::next(int* doc_num, float* doc_score)
12 {
13     int i;
14     int checknext = *doc_num;
15     int searching = 1;
16     int childnext;
17     *QueryNode childQueryNode_[MAX_CHILDREN];
18     float childscore[MAX_CHILDREN];
19
20     while (searching)
21     {
22         searching = 0;
23         for (i=0; i<numChildren_; i++)
24         {
25             childnext = checknext;
26             if (!childQueryNode_[i]->next(&childnext, &childscore[i]))
27                 return 0;
28             if (childnext > checknext)
29             {
30                 checknext = childnext;
31                 searching = 1;
32                 break;
33             }
34         }
35     }
36
37     weightScores(childscore, numChildren_);
```

```

1      *doc_num = checknext;
2      *doc_score = score_function(childscore, numChildren_);
3
4      return 1;
5  };

```

In this example, the MultiAndQueryNode calls the next function of each of its children QueryNode 203 objects in the for loop by iterating over numChildren. Each child QueryNode 203 object gets as input the document number output by another child QueryNode 203 object. This allows for query optimization and for the boolean *and* logic. If the document number returned by a child QueryNode 203 object in the variable childnext is greater than checknext, which was the document number passed to the child QueryNode 203 object, then the child QueryNode's next function located a document satisfying the query model of the child QueryNode 203 object. In this case, the searching flag is set true, and the for loop advances to the next child QueryNode 203 object. Searching is continued until one of the child QueryNodes is unable to locate a document that satisfies the particular search operation represented by the child QueryNode 203 object. The MultiAndQueryNode object then executes a weighting function on the returned document scores. The weighting function is determined by the applications programmer to implement any desired ranking or evaluation of the document scores. The AndQueryNode 203 object sets as its output the document number returned as result of all the child QueryNodes, and a document score based on a scoring function, and returns 1 indicating that the search query was successful.

Referring again to Figure 2, there is also shown the class hierarchy for the NodeCreator 201 class. For each QueryNode 203 class, there is a corresponding NodeCreator 201 class. Each NodeCreator 201 class includes a member function that creates a new QueryNode 203 object of the associated QueryNode 203 class, and variably typed array including both string values ("strings" here includes numerical values if appropriate to the query model) and pointers to other QueryNode 203 objects that stores the parameters to be passed to the new QueryNode 203 object that the NodeCreator 201 object instantiates. The NodeCreator 201 object will pass to the created QueryNode 203 object as its data member the data elements to be operated on by the search operator for the query model. The NodeCreator 203 base class can be represented as follows:

```

35      class NodeCreator {
36          public:

```

```

1         virtual QueryNode* CreateQueryNode (int num_args,
2         argument* args);
3     };
4
5     struct argument {
6         enum {STRING, NODE} type;
7         union {
8             char* STRING;
9             QueryNode* node;
10        };
11    };

```

Each NodeCreator 201 object instantiates a particular QueryNode 203 object, providing it with the proper arguments. The NodeCreator 201 object then returns a pointer to the QueryNode 203 object. Typically, the CreateQueryNode is a call to the object constructor function of the appropriate QueryNode 203.x class. In addition, the CreateQueryNode may perform additional work, such as handling delayed binding to database files for a particular query model.

In some instances, the CreateQueryNode function may not need to call a QueryNode 203 object constructor at all, for example, where there are multiple instances of a given search query in a more complex query such as:
 (OR (AND (HASWORD x) (DATE > 03/03/94)) (AND (HASWORD y) (DATE > 03/03/94)))

Here a DateQueryNode 203.5 object is used twice for a given search query, and instead of creating multiple instances, the CreateQueryNode may return multiple pointers to the same DateQueryNode 203.5 object.

The QueryNode 203 and NodeCreator 201 classes have been described and illustrated as implemented in an object oriented language. Alternatively, the invention can be implemented in a procedural language, such as C, by defining a QueryNode 203 as a variable size struct, with a pointer member to the next function, and similarly by defining a NodeCreator 201 as a struct with a pointer member to the creator function. The invention may also be implemented in assembly language, or by firmware programming in a suitable PLA, FPGA, or similar device.

Referring again to Figure 1 then, for each component part of the search query, the parser 109 determines the search operator keyword 115 and from the search operator obtains a pointer 117 to the particular NodeCreator 201 object that creates the appropriate QueryNode 203 object for the search operator. The parser 109 calls the NodeCreator 201 object, and passes to it an array of strings or QueryNodes 203 objects. The called NodeCreator 201 object will then execute its

1 CreateQueryNode function, by instantiating a new instance of the appropriate
2 QueryNode 203 class, providing it with the passed in arguments. The new
3 QueryNode 203 object will be specifically enabled to perform a search only on the
4 data elements that is has been provided with by the NodeCreator 201 object. The
5 NodeCreator 201 object will then return to the parser 109 a pointer to the new
6 QueryNode 203 object.

7 Because each QueryNode 203 object takes both string and other QueryNodes
8 as arguments, the parser 109 handles complex search queries by creating a "top"
9 QueryNode 203 object that represents the top level search operator, the arguments of
10 the top QueryNode 203 object being pointers to child QueryNode 203 objects, and
11 so on. Thus, in the foregoing example, the parser would create a top AndQueryNode
12 203.1 object. This process is described with respect to Figure 3.

13 First, the search query is input 301 into the search query manager 103, and
14 passed to the parser 109. The parser 109 tests 303 whether there are any search
15 operators in the search query, and if so, proceeds to evaluate the query by reading
16 305 the next search operator and arguments, in this example, the full search query
17 input shown above. The parser 109 tests 307 whether the arguments include any
18 search operators, indicating a nested child search query. If so, the parser 109 recurses
19 and gets 305 the next search operator and arguments. In this example, the arguments
20 are themselves three search operators with arguments. This recursion repeats until
21 the obtained arguments are only string types. In this example, the parser 109 would
22 recurse once, and select the (HASWORD computer) query.

23 At this point the parser 109 evaluates the first child search query (HASWORD
24 computer). The parser 109 looks up 309 the "HASWORD" search operator
25 keyword 115 in the table 113 and finds the pointer 117 to the WordNodeCreator
26 object 201.4. The parser 109 then calls 311 the WordNodeCreator 201.4 object,
27 passing to it an array of one argument of type string, the string data member pointing
28 to the string "computer." The WordNodeCreator's CreateQueryNode function
29 will execute, and create 313 a new WordQueryNode object that describes, through its
30 next function, all documents in the database 121 that contain the word
31 "computer." This WordQueryNode 203.4 object will be stored in the address space
32 of the search application 125. The WordNodeCreator 201.4 object returns a pointer
33 to WordQueryNode 203.4 object to the parser 109.

34 The parser 109 then again tests 303 for remaining search operators. The parser
35 109 then gets 305 the next child search query (NOT (HASWORD Intel)). Since this is
36 itself a complex query, the parser 109 recurses 307 and evaluates the (HASWORD
37 Intel) search query. The parser 109 again looks up 309 the "HASWORD" search
38 operator keyword 115 in the table 113, and calls 311 the WordNodeCreator 201.4 to

1 create 313 a second WordQueryNode object 203.4 for the string "Intel." The parser
2 109 will return from one level of recursion and lookup 309 the "NOT" search
3 operator keyword 115, and obtain the pointer to the NotNodeCreator 201.3 object.
4 It will call 311 this object, passing to it an array with one argument whose type is
5 NODE and whose data member is a pointer to the WordQueryNode 203.4 object for
6 "Intel." The NotNodeCreator 201.3 object will create 313 the corresponding
7 NotQueryNode 203.3 object, returning a pointer to it. The parser 109 will evaluate
8 the (DATE >03/03/1994) conjunct, in a similar manner, creating 313 a
9 DateQueryNode 203.6 object with the arguments of ">" and "03/03/1994."

10 Once all the child search queries are created, the parser 109 will return to the
11 first level of recursion, to complete the evaluation of the "AND" search operator.
12 From the from the table 113 the parser 109 will call 311 the AndNodeCreator 201.1
13 object, passing to it an array of three arguments of type NODE, the array members
14 being the pointers to the QueryNode objects for (HASWORD computer),
15 NOT(HASWORD Intel) and DATE(>03/03/1994) that have just been instantiated.
16 The AndNodeCreator 201.1 object will create 313 a new AndQueryNode 203.1
17 object with its data members being the pointers to the child QueryNodes 203 objects.
18 Figure 4 illustrates the hierarchical structure of the resulting AndQueryNode 203.1
19 object.

20 Once the search query has been completely evaluated by the parser 109, the
21 search application 125 then schedules and manages 315 the execution of the search
22 query as necessary. In the preferred embodiment, execution of the various
23 QueryNodes is delayed until the search application 125 requests the results of the
24 search query.

25 In an alternative embodiment the present invention may be implemented
26 without the use of the parser 109 if the structure of the search query is fixed. For
27 example, if there is always a date, a title, and an author field for every query (or some
28 other fixed format query), then a top AndQueryNode 203.1 or OrQueryNode 203.2
29 object can be created each time with the appropriate child nodes.

30 When the search query is to be executed 317, the processor 101 begins
31 execution with the top node, here the AndQueryNode 203.1 object. Since the
32 AndQueryNode arguments are themselves pointers to further child QueryNode 203
33 objects, the processor 101 will effectively unravel the hierarchical top node,
34 executing 317 the next function of the appropriate child node. The execution order
35 of the various child QueryNodes is not necessarily serial, but can be defined by the
36 applications programmer to optimize search performance.

37 A simple example illustrates the iterative execution of the QueryNode 203
38 objects. Assume a search query of the form

1 AND (HASWORD x) (HASWORD y)

2 and a database 121 including the following documents:

3 Table 1

Document No.	Contents
1	a
2	x
3	c
4	x y

4 When parsed, there will be an AndQueryNode 203.1 top node object with two
 5 child WordQueryNode 203.4 objects. These QueryNodes are not executed until the
 6 search application 125 calls the next function of the top node to return the first
 7 "hit" or non-zero document. Table 2 shows one possible invocation order for
 8 evaluating the search query:

9 Table 2

Invocation	QueryNode	Doc. No. (In)	Doc. No. (Out)	Doc. Score Out
a	WQN (x)	0	2	1
b	WQN (y)	2	4	1
c	WQN (x)	4	4	1

10 Since there has not been any previous search, the search application 125
 11 requests a non-zero document with a document number equal to or greater than 0.
 12 The AndQueryNode 203.1 object then calls (a) its first child WordQueryNode 203.4
 13 object to return a non-zero document, that is a document containing "x," taking 0 as
 14 the input document number. As Table 1 shows, document number 1 does not contain
 15 "x," but document number 2 does, so the WordQueryNode 203.4 object's next
 16 function returns document number 2 as the first non-zero document. The
 17 AndQueryNode 203.1 object then calls (b) the WordQueryNode 203.4 object for
 18 "y", requesting a non-zero document with a document number greater than or equal
 19 to 2. Table 1 shows that document number 2 does not contain "y," nor does
 20 document 3. The next function will iterate, finding "y" in document 4, and return
 21 document number 4. Since the input document number is not equal to the output
 22 document number, there has not been a match to the search query. The
 23 AndQueryNode 203.1 object then calls (c) the WordQueryNode 203.4 object for "x"
 24 again, with an input document number 4, to locate a non-zero document. Here
 25 document number 4 contains "x" and this document number is returned by the
 26 WordQueryNode 203.4 object to the parent AndQueryNode 203.1 object. The
 27 AndQueryNode 203.1 object then returns document number 4 with its document
 28 score to the search application 125.

1 This simple example demonstrates several points. First, the execution of
2 individual QueryNode 203 objects is iterative, each QueryNode 203 object
3 essentially capable of returning a single document number, instead of evaluating and
4 scoring all the documents in the database 121. This allows the search application 125
5 to effectively manage the execution of the search query. Second, most QueryNode
6 203 objects will execute on a limited search space, instead of the entire database 121.
7 In the foregoing example, the WordQueryNode 203.4 object for "y" began its
8 execution on document number 2, document number 1 having already been
9 eliminated from consideration. This allows for efficient searching of the database.

10 In an alternate embodiment, a QueryNode 203 object can execute after
11 instantiation, for example, as part of their constructor function, and compute the
12 scores for documents before the top node's next function is first called. Then the
13 QueryNode 203 object can simply lookup the document scores as when its next
14 function is called. The architecture of the QueryNodes allows this flexibility in
15 design, affording the applications programmer the ability to optimize the performance
16 of each QueryNode 203 subclass.

17 Returning to the first example then, the processor 101 will execute first child of
18 the AndQueryNode 203.1 object, the WordQueryNode 203.4 object for "computer."
19 The next function of this QueryNode is called and either returns 1, and the
20 document number of a document containing the word "computer," or 0 indicating
21 no remaining hits. After the first WordQueryNode 203.4 object returns a non-zero
22 document, the processor 101 then executes 317 the NotQueryNode 203.3 object, and
23 its child node, the WordQueryNode 203.4 object for "Intel," using the returned
24 document number as the starting point for the next function. The result of the
25 WordQueryNode 203.4 object on "Intel" is then passed the NotQueryNode 203.3
26 object, which performs its next function, returning a document number and score for
27 a document that does not contain the word "Intel." Next, the processor 101
28 executes the DateQueryNode 203.6 object with its arguments, returning a document
29 number and score as appropriate. Finally, as all the arguments of the AndQueryNode
30 203.1 object have been evaluated, the processor 101 can then complete the
31 execution 317 of the AndQueryNode 203.1 object's next function, performing a
32 boolean *and* operation on the documents identified by the document numbers
33 returned by the children nodes. If the result is a match of the document numbers, it is
34 returned to the search application 125 for outputting on the display. Depending on
35 user inputs in response, the search application 125 may locate another document, or
36 terminate the search.

37 Referring now to Figure 5, there is shown the flowchart for creating new
38 QueryNodes 203 subclasses to integrate new query models into the information

1 retrieval system 100. First, the applications programmer derives 501 a new
2 QueryNode 203.x subclass from the QueryNode 203 base class in a conventional
3 manner, inheriting the base class data members, specifically the next function, and
4 defining the appropriate number and type of arguments. The applications
5 programmer may then further specify 503 the next function for the new subclass if
6 desired to optimize its performance. A particular scoring function can then be
7 defined 503 appropriate to the query model. To provide a user interface for the
8 query model, a search operator keyword is defined 507. The applications
9 programmer then derives 509 a new NodeCreator 201.x subclass for the new
10 QueryNode 203.x subclass. This NodeCreator 201.x subclass will create a new
11 instance of the QueryNode 203.x subclass as described. Finally, the applications
12 programmer updates the table 113 by storing 511 the search operator keyword with a
13 pointer to the new NodeCreator 201.x object. When the search application 125 is
14 compiled, the table 113 will have the correct addresses for the various NodeCreator
15 201 objects. The parser 109 will then be able to instantiate a QueryNode 203.x
16 object for the new query model when the appropriate search operator is used in a
17 search query.

I claim:

1 1. A system for executing search queries on a database comprising a plurality
2 of documents, each search query having a search query operator and at least one
3 data element, the system comprising:

4 a search query manager for receiving a search query including at least one
5 search query operator and at least one data element to be operated on the
6 search query operator;

7 a storage device for storing associations between search query operators
8 and query node creators, each query node creator for creating a query node to
9 execute a search query for a search query operator upon at least one data
10 element, the created query node for executing a search query on the database
11 to identify a document therein corresponding to the data element of the search
12 query, and returning an identifier of the document, and a score for the
13 document;

14 a parser, coupled to the storage device, and further coupled to the search
15 query manager and receiving therefrom the search query, the parser
16 identifying each search query operator and data element in the search query,
17 and for each search query operator in the search query, identifying the query
18 node creator associated with the search query in the storage device, and
19 calling the associated query node creator to create a query node for the search
20 operator and the data element; and

21 a processor, coupled to the parser, and receiving therefrom a first query
22 node, and executing the first query node.

1 2. The system of claim 1, wherein each query node further comprises:

2 an argument list containing a first document number;

3 a code object for receiving the argument list, and searching the database to
4 retrieve a document having a second document number greater than the first
5 document number and a non-zero document score, the code object setting the
6 first document number to the second document number and a document score
7 to the non-zero document score.

1 3. The system of claim 1, wherein the first query node is comprised of a
2 plurality of nested query nodes, the processor executing the first query node by
3 executing each nested query node.

1 4. In a system including a database of documents, each document having an
2 identifier and a document score, and a processor for executing search queries on the

3 database to retrieve selected documents, each search query including at least on
4 search query operator and at least one associated data element, a memory comprising:
5 an application program for executing search queries on the database,
6 including:
7 a query node base class including:
8 a search data element member for storing a search data element; and,
9 a search function member accepting as input a first document
10 number, the search member function executable by the processor for
11 searching the database to retrieve a document having a second
12 document number greater than the first document number, and a non-
13 zero document score, and to return the non-zero document score and
14 the second document number;
15 at least one query node derived from the query node base class;
16 a query node creator base class including:
17 a query data element member for storing at least one query
18 data element including either a search data element or a
19 subordinate query node;
20 the query node creator base class returning a superordinate
21 query node having as a search data element the query data
22 element; and
23 at least one query node creator derived from the query node
24 creator base class; and
25 a dynamically allocated storage structure for storing derived
26 query node creators in association with search operators; and
27 a parser having a parsing member function for parsing a search
28 query to identify each search operator and associated data elements
29 therein, and coupled to the storage structure for determining a query
30 node creator associated with each search operator, the parser calling the
31 query node creator associated with each search operator to creator a
32 query node, and passing to the query node creator the data elements
33 associated with the search query operator.

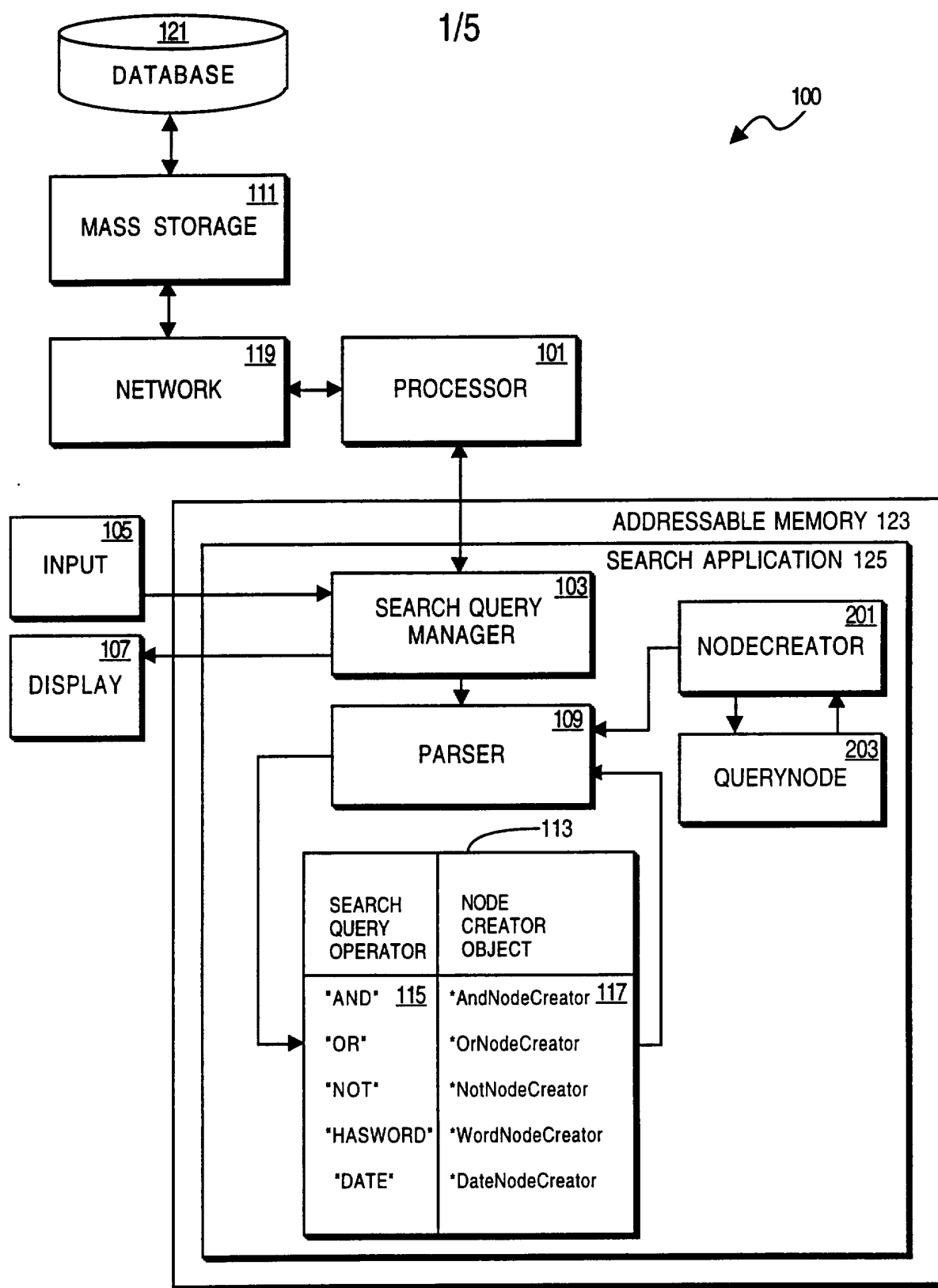


FIGURE 1

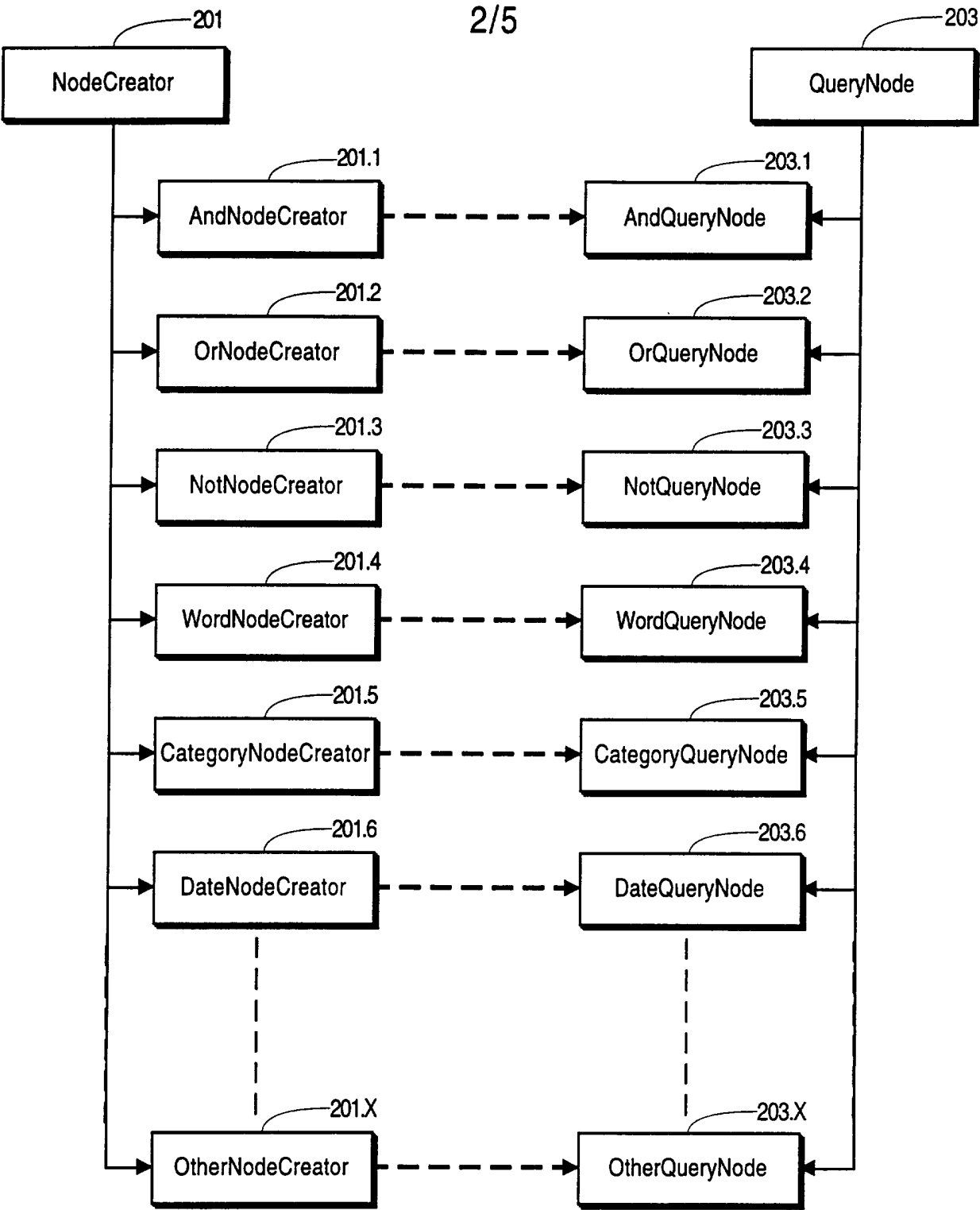
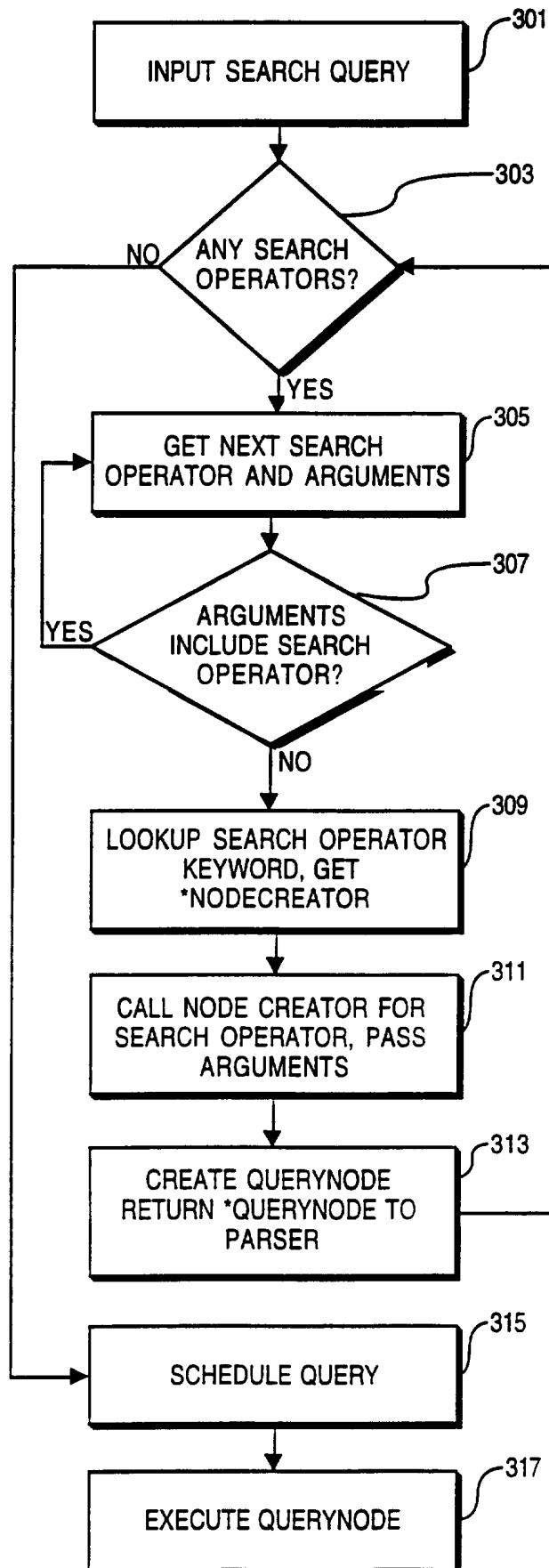


FIGURE 2

3/5

FIGURE 3

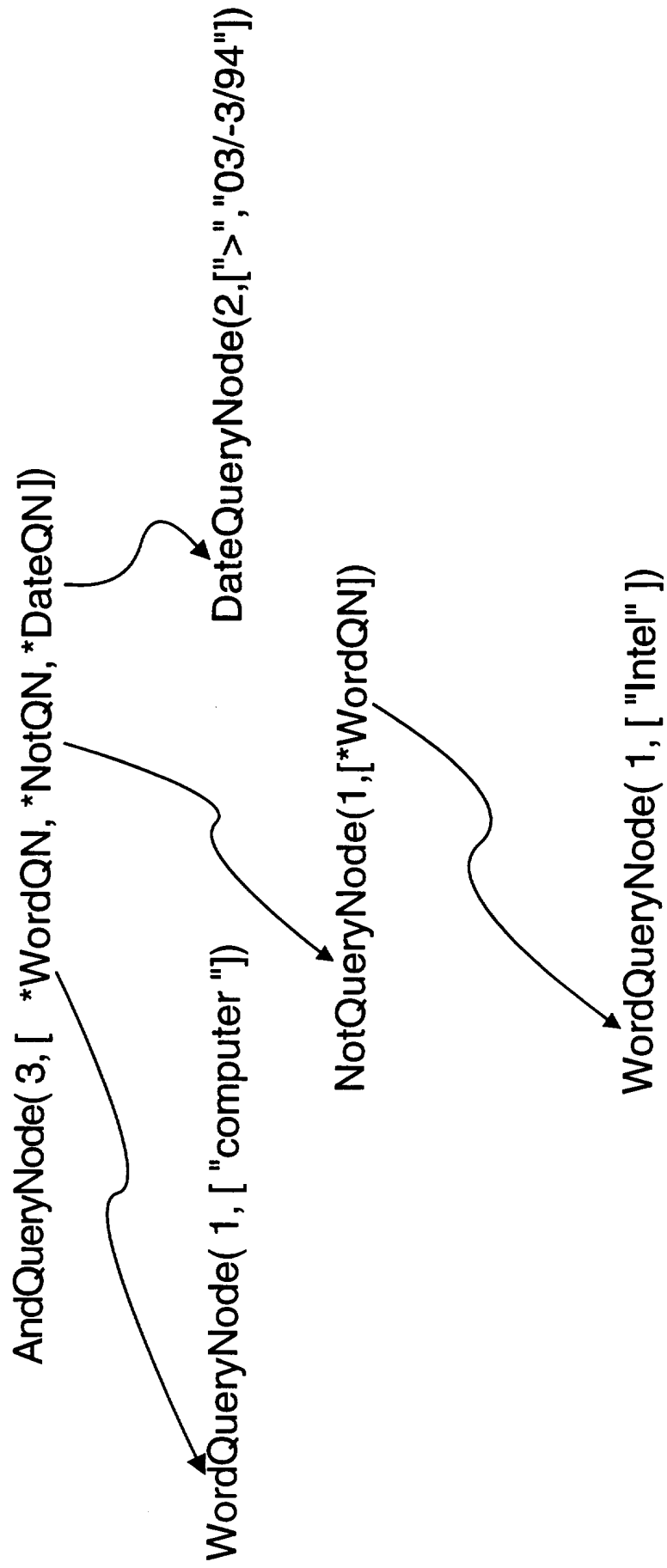


FIGURE 4

5/5

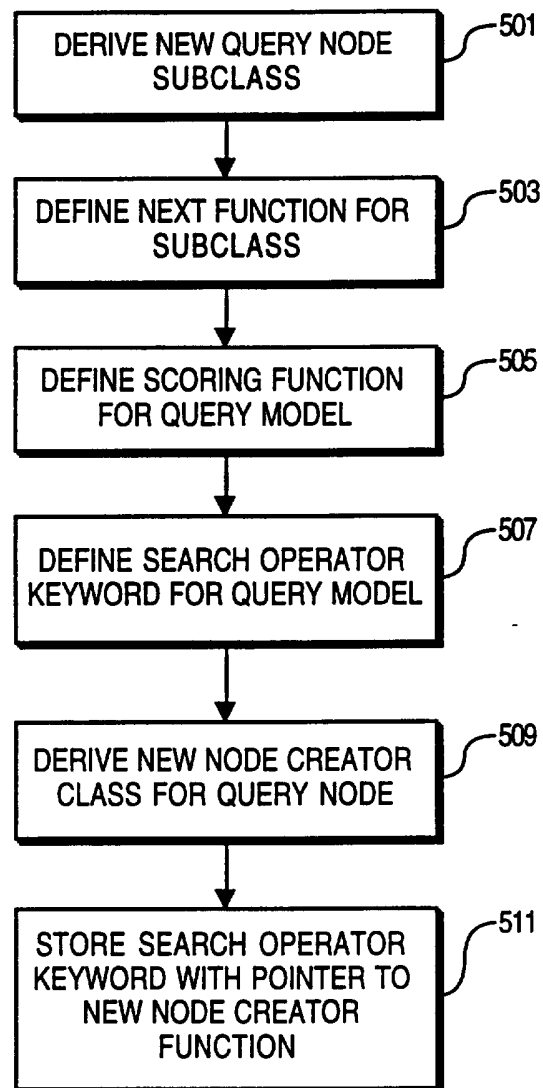


FIGURE 5