An Intelligent User Interface for Browsing Satellite Data Catalogs

Robert F. Cromp and Sharon Crook
Science Applications Research, Inc.
National Space Science Data Center
NASA/Goddard Space Flight Center
Greenbelt, MD 20771

Abstract

A large scale domain-independent spatial data management expert system that serves as a front-end to databases containing spatial data is described. This system is unique for two reasons. First, it uses spatial search techniques to generate a list of all the primary keys that fall within a user's spatial constraints prior to invoking the database management system, thus substantially decreasing the amount of time required to answer a user's query. Second, a domain-independent query expert system uses a domain-specific rule base to preprocess the user's English query, effectively mapping a broad class of queries into a smaller subset that can be handled by a commercial natural language processing system.

The methods used by the spatial search module and the query expert system are explained, and the system architecture for the spatial data management expert system is described. The system is applied to data from the International Ultraviolet Explorer (IUE) satellite, and results are given.

1 Introduction

Large amounts of data from a variety of sources covering a multitude of domains are stored in databases. Increasingly, these databases are becoming more complex and intertwined. In addition, research involving this data often is multidisciplinary in nature, requiring the researcher (scientist or manager) to access multiple databases over a network of non-homogeneous computer systems. Users are investing their important time in learning and recalling how to use these systems. Unless a person has a pressing need to know an answer, a system is not learned—the start-up time for learning how to use the system outweighs the importance of the answer.

The Intelligent Data Management (IDM) Project at the National Space Science Data Center (NSSDC), NASA/Goddard Space Flight Center, has been involved over the past few years in investigating, developing, and proving methods which facilitate the accessing of databases for the user [4, 15]. Our research has concentrated on combining specially developed software with commercial packages, and applying our systems to non-trivial domains.

The general domain-independent methodologies which the IDM research has produced define the steps that are necessary for automatically developing an intelligent user interface to access databases. This paper reports on our approach to creating a spatial data management expert system that serves as an intelligent front-end to databases that contain spatial data. We believe this is an important step in designing a system that researchers can use to access the voluminous amounts of data that will be produced by on-going and future NASA missions, such as IUE, Space Station, Space Telescope and Eos.

2 The Spatial Data Management Expert System Architecture

We have designed and implemented a large scale domain-independent spatial data management expert system that provides answers to users' English queries. The system integrates our in-house developed modules with various commercial products.

Figure 2.1 shows the conceptual arrangement and data flow of the system. A user enters his query in English, and optionally, by means of a graphics interface, selects a set of spatial regions to which he wants to restrict his query. The English portion of the query is passed through a query expert system that uses a domain-specific rule base to preprocess the query for handling by DataTalker, a commercial natural language database front-end marketed by Natural Language, Incorporated [13]. The query expert system channels a broad class of queries into a smaller set which DataTalker is then able to process. In addition, the query expert system passes the region coordinates to a specially designed spatial search module that returns a list of the primary keys of the records in the database that fall within the selected regions. DataTalker is passed an English query that is formed from the user's transformed query and the set of primary keys.

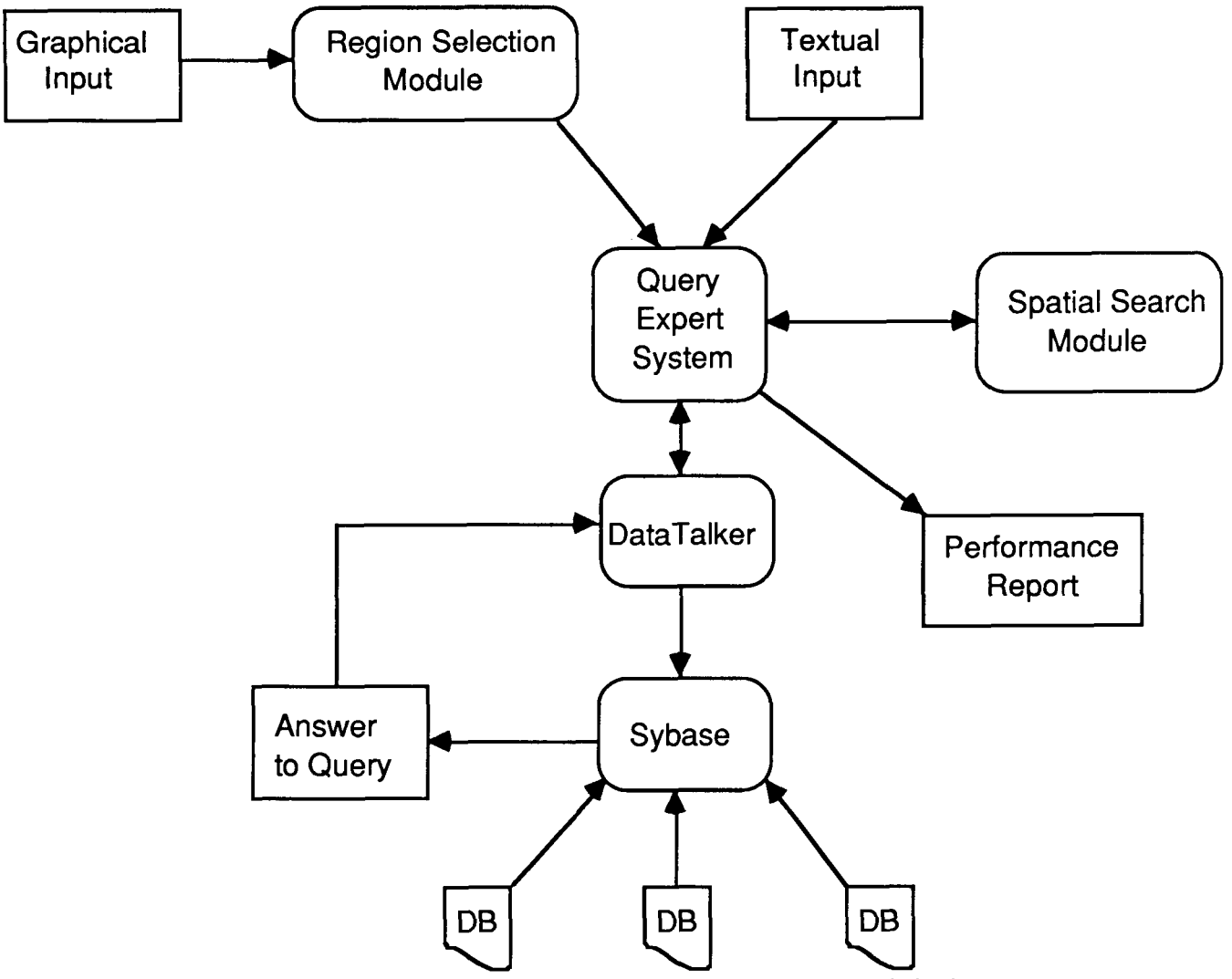

Figure 2.1: The overall system architecture for the spatial data management expert system.

When DataTalker is installed on a machine, it is connected to a database management system. DataTalker automatically invokes the DBMS whenever it successfully parses a transformed query. The low-level database accessing is transparent to the user, and the format of the query results is controlled by DataTalker. If DataTalker cannot parse the query (see Section 4), it returns a message to the query expert system indicating so.

The Spatial Data Management Expert System causes DataTalker to store its result in a file. The contents of this file are used to update a text window of query results on the user's console.

If the user is not satisfied with the way his query is transformed or answered, he can file an electronic performance report which is later reviewed by the

domain expert. The domain expert uses this information to guide the evolution of the system with respect to the types of queries asked by the users.

3 Spatial Search

In order to answer queries about spatial data, we must be able to access the information associated with those records quickly. A database is too slow for this problem forcing the development of quicker methods for accessing data. We chose quad trees and k-d trees as suitable data structures for answering spatial queries about records. The implementation of these data structures allows us to perform such tasks as finding the nearest neighbor to a given coordinate and determining all records within a certain region. After finding the answer to such spatial queries, we can return the appropriate primary database keys to the expert system which uses the keys to gain any other necessary information from the database.

3.1 Tree Construction

Our implementation of these two data structures relies on the spatial relationships among records based on coordinate values for a two dimensional space, for example right ascension and declination. We use similar methods to build each of these two types of trees from a set of n records, $\{x_1, x_2, \ldots, x_n\}$. In order to create a quad tree, we choose a point, x_i, from this set. This point determines the root node of the tree and divides the space into four quadrants based on the coordinate values of that point. Each quadrant is represented by one of the four subtrees extending from the four children of x_i. We continue to divide the space in this manner until each record is represented by a node in the quad tree. Figure 3.1 shows an example of the construction of a quad tree from the data in Table 3.1.

Table 3.1: Sample observation data (simplified).

Name	Right Ascension	Declination
Algol	3	41
Altair	20	9
Bellatrix	5	6
Edasich	15	59
Regulus	10	12
Sheratan	2	21

The creation of the k-d tree differs slightly in that the point chosen from the set of records is used to divide the space into two subspaces which determine

two subsets of the set. This process continues in alternating dimensions until each record is represented by a node in the k-d tree. Figure 3.2 depicts this process.

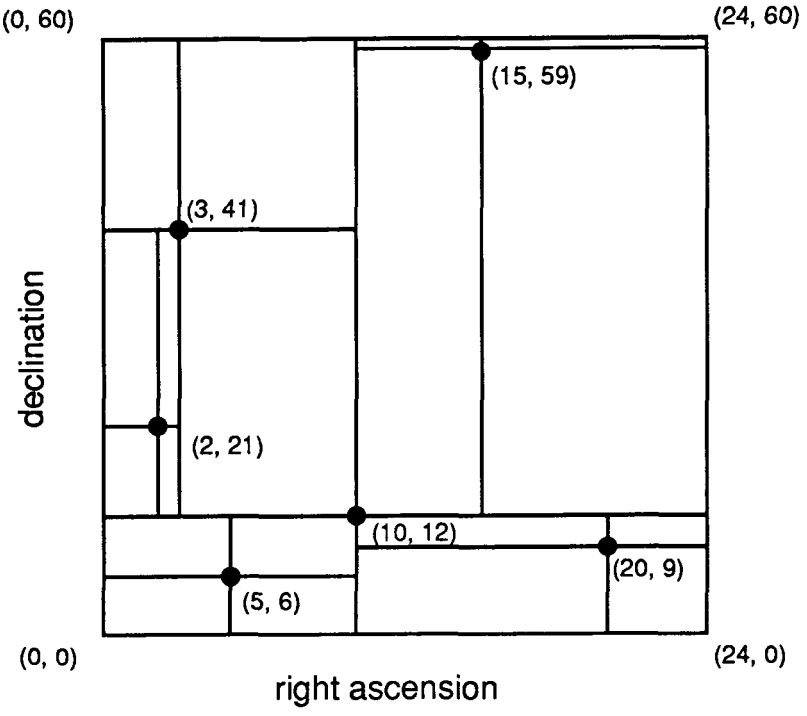

Figure 3.1: The structure of a quad tree.

We chose to store the IUE observation data in the internal nodes of the trees due to the large number of records and the fact that deletions are not needed for our application. If node deletions had been necessary, we would have chosen pseudo quad trees and pseudo k-d trees for implementation. These structures store data associated with points only at the leaf nodes of the tree providing the means for a large number of efficient deletions and insertions [14]. An observation catalog continues to grow as long as the satellite which supplies its data continues to function. For this reason we needed the capability to insert a small number of observations after the completion of the trees, which is possible and efficient at the leaf level if we allow a slight depth increase.

Due to the nature of these data structures, we developed an object oriented implementation in C++ [16]. The key structure in C++ is a user-defined type called a class. In our application classes include such objects as nodes, lists of nodes, trees, and regions in space. This object oriented approach is particularly well-suited to this problem due to the hierarchical nature of these structures.

Another advantage of such an approach is the ability to create programs which are concise, easy to understand, and highly maintainable.

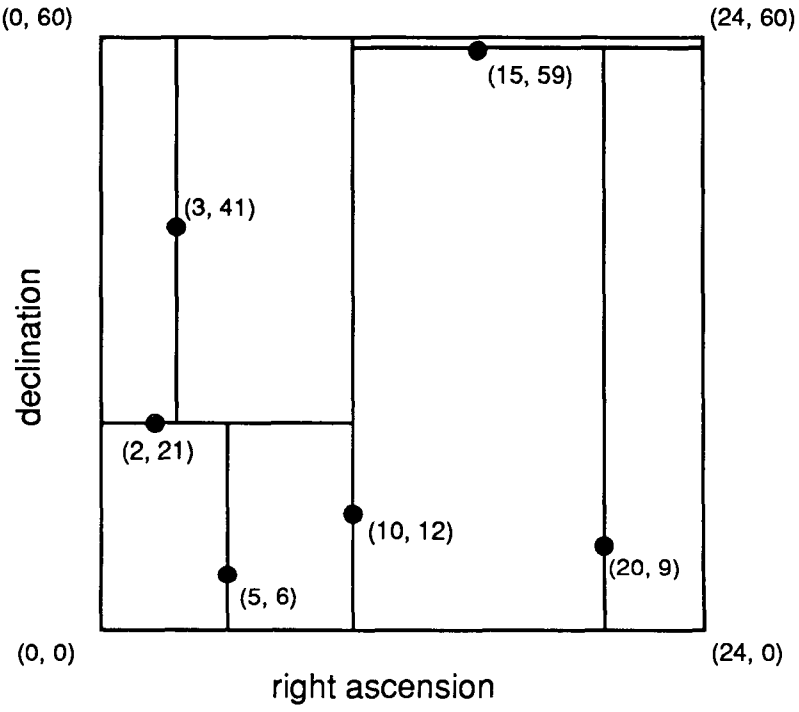

Figure 3.2: The structure of a k-d tree.

In order to minimize search times for the quad tree and k-d tree, we chose to expend more effort in building the trees. By allowing more time for pre-processing, we can optimize the trees through balancing. This is accomplished by imposing the condition that for every node x_i in the tree, every subtree of that node may contain no more than half of the nodes whose root is x_i. This is easily done by choosing the proper point in each observation subset when placing nodes in the tree. At each step in the recursive process we must select the median point in either dimension. Thus all remaining observations in that subset will be equally divided with respect to the point chosen meeting the condition that no subtree of the new node may contain more than half the nodes in that subset. This optimization insures that the maximum path length in a tree of n nodes is the floor function of $(\log_2 n)$ and the maximum total path length is $\sum_i \text{floor}(\log_2 i)$ [6].

There are various quick algorithms for selecting the kth element of a set of n elements and partitioning the set about that element. These provide efficient

methods of determining a median and implementing the process described above. One algorithm due to C. A. R. Hoare (referenced in [2]) performs the task in O(n) average time and is easy to implement. This allows us to create an optimized tree in $O(n\log_2 n)$ time. We chose this algorithm for its simplicity and efficiency. Another more complicated algorithm due to Floyd and Rivest uses only n+k+o(n) comparisons and runs very quickly [7].

When we wish to create a tree, we use the standard C++ **new** command to allocate a large block of virtual memory which will store the tree nodes. After creating the tree from the data in an observation catalog, we can send all of the information needed for rebuilding the tree to a file. This information includes the right ascension and declination values, the primary keys of the database relation, and the addresses of the children for each node in the tree. We subtract the base-address from these addresses in order to save the general tree structure which is independent of its location in memory. We can rebuild the tree using this data by adding the node address values to the base-address for a newly allocated block of memory. With this ability, the preprocessing time necessary for initially creating and balancing a tree is eliminated. Trees are always readily available and efficiently recreated.

3.2 Spatial Queries

The simplest type of query which can be answered by searching a quad tree or a k-d tree is what Knuth calls a "simple query" [10]. This involves searching a tree for a specific record using a point search based on the spatial data. A simple algorithm performs the search by making a comparison at each node and choosing the correct subtree for the next test. The average time required is proportional to the total path length divided by the number of nodes in the tree [6]. Thus no simple search will require more comparisons than the maximal path length.

A more general type of spatial query is a "region query" [10] in which we specify a set with which records must intersect. This class of query might be invoked by a question such as "List the primary keys of all observations with right ascensions between 8 hours and 16 hours and declinations between 20 degrees and 50 degrees." Figure 3.3 shows an example of such a region in space.

The conventional method of answering a rectangular region query by searching a quad tree involves a recursive procedure [6]. An almost identical algorithm will suffice for k-d trees [1]. Similar procedures can be defined for different types of geometric figures such as the shape of a satellite camera aperture.

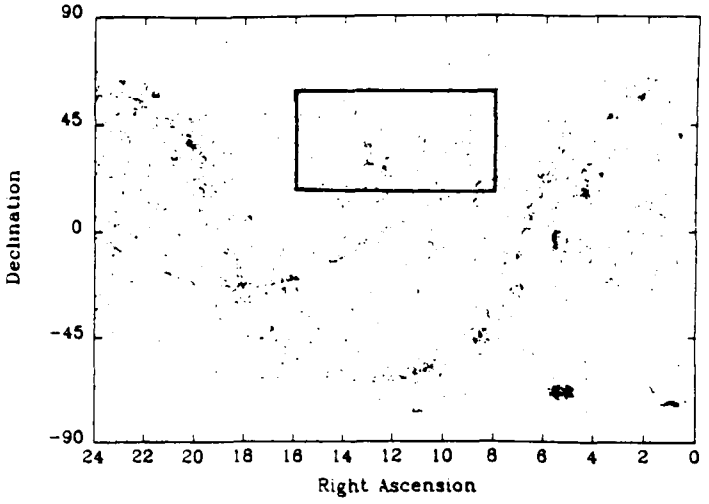

Figure 3.3: An example of a region to be searched.

The last type of query necessary for our application is a nearest neighbor query. Given spatial coordinate values, it is often desirable to find the nearest observation to that point. Bentley provides a complicated algorithm for performing this process in k-d trees for which empirical tests show $O(\log_2 n)$ time [1]. A slight variation due to Friedman, Bentley, and Finkel was *proven* to be $O(\log_2 n)$ [8]. For simplicity, we implemented a more naive algorithm for finding a nearest neighbor in either type of tree which relies on the region search described above. Future empirical tests are planned in order to determine the performance of our algorithm.

4 Natural Language Processing

There are a small number of commercially available natural language processing packages. We have found DataTalker, by Natural Language Incorporated, to be a very powerful system for handling English queries. DataTalker can communicate with most of the major database management systems. The user's query is converted into the necessary database query language, the DBMS is invoked, and DataTalker displays the query result to the user in a structured format. If DataTalker is unable to parse the user's query, a message is generated explaining why. A query that is syntactically incorrect, contains metaphorical expressions, or consists of specialized vocabulary which DataTalker has not been taught causes an error message to be produced.

DataTalker must be configured to the domain. The person who does this configuration must be familiar with both the domain and the structure and contents of the database. We refer to this person as the domain expert. Preferably, this person will have a list of questions that the user base might ask.

Since we have had limited experience with DataTalker (this was our first use), we aren't qualified to report on the amount of time it takes to configure the system for some domain. However, we will note that the current version of DataTalker (version 3.0) is much friendlier than its previous versions, and we find that it is easy to build a configuration in incremental steps.

5 The Query Expert System

DataTalker was designed to be used solely interactively. This is unfortunate, because many different systems could benefit from its capabilities if an interface existed that allowed it to be included as a component. We found a natural language interface was desirable for many of the domains with which we were working, so we faced the choice of either circumventing DataTalker's limitations, or designing our own natural language processing system. We quickly adopted the former option, since the latter requires a long-term effort into computational linguistics if a robust, mature system is sought [9,11].

If a person submits a query to DataTalker that the system is unable to understand, a short message is returned which may or may not indicate the problem. The person is given immediate feedback and is expected to rephrase his query so that the system can answer it, or abandon it because it is obviously beyond the scope of DataTalker. If the system is able to parse the query, render it in the necessary database query language, and successfully query the database, then the result is returned to the user. In any case, the person quickly realizes whether his query was handled properly.

When DataTalker realizes that it is unable to handle a query, it returns a brief message indicating the problem, possibly preceded by its English interpretation of the query. The message typically starts with a phrase such as "Sorry, ... ," "The database contains no information on ... ," or "I don't know ... ."

There is also a possibility that DataTalker misinterprets the user's query and provides what it believes to be a correct answer, although in reality the answer corresponds to a question different from the one the user asked.

The query expert system (QES) is a module we designed and implemented in LISP which serves as an intelligent front-end to DataTalker. It accepts the user's query, transforms it according to a domain-specific rule base, and passes it on to DataTalker. If DataTalker replies that it does not understand the query, then QES packages the original query along with the transformations it underwent and DataTalker's response, and files this. The domain expert periodically reviews this file, and makes changes to the rule base, if possible, so that QES can transform the problematic queries into queries that can be parsed by DataTalker. This feature allows the system's performance to improve throughout its lifetime.

5.1 The Textual Rule Base

The domain expert creates a text file that contains the rules and functions which are used to transform the user's query prior to passing it to DataTalker. The grammar for representing the textual rule base is shown in Figure 5.1.

<*rule base*> ::= <*rule*> <*rule base*> | <*function*> <*rule base*> | ε
<*rule*> ::= <*pattern*> '==>' <*transform*> {<*explanation*>}
<*pattern*> ::= <*pattern head*> <*pattern tail*>
<*pattern head*> ::= <*word*> | <*number matchup*> | <*word set*>
<*pattern tail*> ::= <*pattern item*> <*pattern tail*> | ε
<*pattern item*> ::= '...' | <*word*> | <*number matchup*> | <*word set*>
<*word*> ::= <*ascii character excluding whitespace*>$^{+}$
<*number matchup*> ::= '#'{<*digit*>}*
<*word set*> ::= '{' <*word*> <*word list*> '}'
<*word list*> ::= ',' <*word*> <*word list*> | ε
<*transform*> ::= <*transform item*> <*transform*> | <*transform item*>
<*transform item*> ::= <*word*> | <*function call*>
<*function call*> ::= <*function name*> '(' <*parameter list*> ')'
<*function name*> ::= <*letter*> <*ascii character excluding whitespace*>$^{+}$
<*parameter list*> ::= <*number matchup*> <*number matchup list*>
<*number matchup list*> ::= ',' <*number matchup*> <*number matchup list*> | ε
<*function*> ::= <*function call*> '=' { <*function body*> | '(' <*expression*> ')' }
<*function body*> ::= { <*condition*> ',' <*action*> <*carriage return*> }$^{+}$ <*blank line*>$^{+}$
<*condition*> ::= <*simple comparison*> | <*condition*> **'and'** <*condition*>
<*simple comparison*> ::= <*operand*> <*relational*> <*operand*> { <*relational*> <*operand*> }
<*operand*> ::= <*number matchup*> | <*number*>
<*relational*> ::= '<' | '>' | '=' | '<=' | '>=' | '<>'
<*action*> ::= '!' | { <*word*> | '(' <*expression*> ')' }$^{+}$
<*expression*> ::= <*mathematical expression possibly involving parameters to function*>

Figure 5.1: The grammar for defining domain-specific rules and functions.

If DataTalker is unable to handle a query in one form, but can successfully answer the same query when it is reworded, then one or more rules can be added to the rule base so that QES can automatically transform the misunderstood

query prior to passing it to DataTalker. Application of a rule transformation causes a portion of the query to be restructured, deleted or replaced by text specified by the rule. QES continually applies its domain rules to the successive renderings of the query until no rules are activated. The final form of the query at this point is displayed to the user along with explanations of why various transformations were used. The user must then consent that the altered version of the query is the closest match between his original query and the information directly obtainable from the database, or else the query is not sent to DataTalker. If the user is unsatisfied with the way his query has been altered, then he is allowed to file an electronic report stating his reasons. This, along with the QES actions for the query, can later be reviewed by the domain expert leading to possible enhancements to the rule base, or clarification back to the user.

A query transformation rule consists of a pattern and the transform which is to occur given that the pattern is encountered in the query. An optional explanation can also be supplied with the rule. This explanation is displayed to the user if this rule is used to alter his query.

A pattern is a series of pattern items. The possible pattern items are **...**, *<word>*, *<number matchup>*, and *<word set>*. The pattern item **...** matches zero or more consecutive words of the query. The item *<word>* must match the query word exactly. A *<number matchup>* matches the query word if it is a number. A *<word set>* is a list of words, and a match occurs if any member of this set is the same as the query word. An implicit **...** starts and ends each pattern. Each word in the query must be accounted for by some item of the pattern in order for the pattern to match the query. In addition, if item *i* directly precedes item *j*, then the portion of the query matched by item *i* must directly precede the portion of the query matched by item *j*. Figure 5.2 shows the match between a query and a pattern.

QUERY: show all the observations taken at the 2175 A bump
PATTERN: ... observations ... {at, of} the # A bump ...

Figure 5.2: The match-up between a query and a pattern.

In a rule, the transform describes how the query should be altered provided the pattern matched it. A transform consists of a combination of one or more function invocations or words. The substitutions in the query that occur depend on the make-up of the transform. The ordering of words in the new query is based on the pattern. All **...** pattern items are incorporated unaltered into the new form of the query. The left-most *<word>* pattern item that also occurs in the transform is replaced by the entire transform in the new query. If the

transform consists solely of a function invocation, then the left-most number in the pattern that is bound to one of the function's parameters is replaced by the value returned when the function is evaluated. All other query words and numbers that are matched to *<word>*, *<word set>*, or *<number matchup>* pattern items are deleted. Figure 5.3 shows the effects of a rule firing on a query.

QUERY: how many observations were made at high resolution
RULE: observations ... {at, with} high resolution ==> high dispersion observations

Pattern Item	Query Word	Contribution to New Query
...	how many	how many
observations	observations	high dispersion observations
...	were made	were made
{at, with}	at	
high	high	
resolution	resolution	
...	ε	

NEW QUERY: how many high dispersion observations were made

Figure 5.3: How a rule alters the structure of a query.

The domain expert can also define functions that can be used in the rule's transform. Figure 5.4 is one example of a function and a rule that uses that function. A function definition consists of the function name, a list of its parameters, and the function body. The body can be either a mathematical expression or a collection of condition-action pairs. A function that is defined as a straight mathematical function returns the result of evaluating that function under the current function parameter bindings. If the body consists of condition-action pairings, then if a condition is found that holds true for the given parameter bindings, the action is substituted in place of the function call in the rule's transform. If an action has an expression embedded within it, that part is replaced by its evaluated form. The cut action "!" has special significance. Conditions whose action is "!" are evaluated prior to all other conditions. If a condition holds whose action is "!", the rule which invoked this function is not applied for the current situation. Figure 5.5 shows two rules that could both potentially fire for a given query, even though their results are quite different. The cut ensures that the correct rule is applied.

When a *<number matchup>* pattern item matches a number in the query, a binding occurs such that all occurrences of that *<number matchup>* token that appear in the transform are replaced by the number it matched from the query. If more

```
RULE:  observations ... at the # A bump ==> f(#) observations

f(#)  = 1000 <= # < 1920, short wavelength
        2000 < #, long wavelength
        1920 <= # <= 2000, short wavelength or long wavelength
        # < 1000, !
```

Figure 5.4: An example of a rule which relies on an expert-defined function. The rule does not apply if the *<number matchup>* token is bound to a value less than 1000.

than one *<number matchup>* item occurs in a rule's pattern, the *<number matchup>* tokens should be distinct from one another. If the expert wants to write a rule that fires only if, say, the same number appears twice in the query, then the pattern should use two different *<number matchup>* tokens, and the transform should invoke some function which checks that the bindings of the two tokens are equal in value.

```
RULE 1:  observations ... at # A ==> f(#) observations
RULE 2:  observations ... at # A ==> g(#) observations

f(#)  = 1000 <= # < 1920, short wavelength
        2000 < #, long wavelength
        1920 <= # <= 2000, short wavelength or long wavelength
        # < 1000, !

g(#)  = # < 4, high dispersion
        # >= 4, low dispersion
        # > 10, !
```

Query	Transformed Query (Rule used)
show the observations taken at 1 A	show the high dispersion observations (2)
show the observations taken at 1800 A	show the short wavelength observations (1)

Figure 5.5: Use of a cut "!" can ensure the correct application of a rule.

5.2 Compiling the Rule Base

The textual rule base must be compiled before QES can use it. We have developed a LISP module that parses and compiles the expert's rule base. Compilation must occur any time the textual rule base is altered if QES is to make use of these changes. Four structures are computed during compilation: the rules, the functions, the word occurrences, and the rule thresholds.

Each rule is encoded as a list consisting of a unique rule number, the pattern and the transform. The pattern is represented as a list of the pattern items, in string format, where the *<word set>* pattern item is structured as a list of

strings. The transform is rendered as a list of its components. A function call is stored as a list whose head element is the function name and whose remaining elements are the parameters to be passed.

Each function is represented as a list consisting of the function name, a list of its parameters, and an evaluatable LISP encoding of the function body. If necessary, the condition-action pairings are reordered so that the potential cuts are evaluated prior to the other clauses. If a mathematical expression occurs in the function body, then code is generated which converts the result to a string automatically so that it can be concatenated into the action or template correctly. Figure 5.6 shows the encoding for one of the functions from Figure 5.5.

```
("f" ("#")
     (cond  ((< "#" 1000) "!")
            ((and (<= 1000 "#") (< "#" 1920)) "short wavelength")
            ((< 2000 "#") "long wavelength")
            ((and (<= 1920 "#") (<= "#" 2000)) "short wavelength or long wavelength")))
```

Figure 5.6: The compiled form of an expert-supplied function. The function body can be evaluated once the value for the parameter is substituted.

A word occurrence index is also computed and placed in the compiled rule base. An element of this index consists of a word and a list of those rules in which that word appears, and the number of times it appears in each of those rules (only the rule's pattern is used, not its transform). Instead of indexing specific numbers, the element "#" appears in the index along with a list of the rules which contain numbers and the number of occurrences in each rule. An entry is made for each word in a *<word set>* pattern item, whereas the **...** pattern item is ignored. The word occurrence index is used during run-time to increase the speed of the inference engine.

The last structure that is placed in the compiled rule base is referred to as the rule threshold list. An element of this list consists of a rule number and the number of items in that rule's pattern, excluding **...** pattern items. This list is also used by the inference engine in its attempt to find the correct rule to apply.

5.3 The Inference Engine: Applying the Rule Base

The QES inference engine uses the query to decide which rules are potentially applicable for transforming it. The method of using the data to drive the inferencing is referred to as forward chaining.

The size of the rule set is most likely much larger than the number of words in the user's query. The user would experience a noticeable wait if the inference engine had to inspect each rule every time it was seeking to transform the query. Also note that there are relatively few rules applicable to the query at any given time. The inference engine must be implemented so that it minimizes the number of rules it attempts to apply for a given query.

Each rule starts with a score of zero. A list is formed of the unique words that are found in the query and their number of occurrences. Each word in turn is taken from this list, and is used with the occurrence index to find those rules in which it occurs. The score for each of these rules is incremented by one for each time the word occurs in its pattern, not exceeding the number of times the word occurs in the query.

All rules whose score equals or exceeds their rule threshold value are placed in the conflict set. These are the rules that can potentially affect the given query at this time. All the other rules cannot apply, either because the query is shorter than their patterns, or their patterns contain one or more words which are not in the query. The rules found in the conflict set may or may not apply. Even though their patterns contain the appropriate words, the words could be in a different order, or a function invocation in their transform could return a cut "!".

More than one rule in the conflict set may satisfy all the conditions for firing. However, one rule may fit the circumstances better than another because it is more specific [3]. To ensure that the "best" rule is found and applied as quickly as possible, the rules in the conflict set are ordered according to their specificity. A rule is considered more specific than another if its pattern contains more items. Rules with longer patterns are tried before ones with shorter patterns. A rule is automatically removed from the conflict set if it is unsuccessfully applied. Once a rule is found that can fire, a new query exists, and the conflict set must be recomputed anew.

In general, a given query undergoes a number of transformations until the inference engine is unable to find any rules which can alter it. This is known as a quiescent state and corresponds to an empty conflict set. Once this state is achieved, the final form of the query is returned, it is displayed to the user, and QES passes it to DataTalker. We must make sure that a quiescent state is reached. In particular, we must guard against a collection of rules which bounces the query back-and-forth. To remove the possibility of infinite looping, the principle of refraction [12] is used. A rule is removed from the conflict set if it has already been applied to the query in its current form. Most likely, if infinite recursion accidentally was introduced to a rule base, only a handful of

rules would be involved. This method of detecting infinite looping works for any number of rules.

6 Results

To test our spatial data management expert system, we have selected a subset (about 5,000 records) of the IUE Minilog (about 70,000 records). After interviewing an astrophysicist who is an IUE expert, we developed a LISP program that converted the magnetic IUE Minilog tape data into a file consisting of the various fields available (e.g., right ascension) or computable (e.g., continuum level) for each observation.[1] The resulting file of IUE observations was ingested into Sybase, a commercial DBMS available from Sybase, Inc. [17].

DataTalker was configured as far as possible on the IUE domain, and a rule base for QES was generated which contains about 50 rules, some of which have been used as examples throughout this paper. Figure 6.1 contains a list of some of the questions our system can answer, where italics indicates that QES performs some modification to that part of the query before it is passed to DataTalker.

What observations *of HD112244 have been taken*?
What high dispersion IUE observations *of HR4908 have been taken*?
What observations *around 1260 A* have been taken of CPD -56 5498?
What observations of O-stars have been made *at 0.1 A resolution at 2600 A*?
What low dispersion observations have background levels *larger then* [sic]*100 DN* and continum [sic] levels less than *220 DN*?
What low dispersion observations have been made *from 1150 - 3000 A*?

Figure 6.1: Actual user queries that can be successfully answered by our spatial data management expert system. Italicized portions of the query are modified by QES prior to being passed to DataTalker.

By using our region selection and spatial search modules, any of the questions can be restricted to a smaller portion of the database. We believe this increases the speed of the system because our method of performing the spatial search is significantly faster than relational database search methods that use non-spatial primary keys to construct their trees. The quad and k-d trees take full advantage of the numerical properties inherent in the spatial data, whereas the trees constructed by DBMS's must be searched entirely to retrieve all records which meet some spatial criteria. Our spatial data management expert

[1]We are currently developing a method to perform this database conversion process automatically so that a domain expert can do it without requiring a programmer to help. A lot of computation can be saved, and more interesting questions can be answered, if the database is structured with respect to the types of questions the users ask.

system presents a list of the primary keys to the DBMS immediately, thus eliminating the need for the database to be exhaustively searched.

7 Future Research

This is the first large scale domain-independent spatial query management expert system that we have built. There are many research topics to be examined as we continue to develop our system to meet the users' needs. We mention some major areas for improvements or further investigation.

- As mentioned earlier in the paper, our nearest neighbor search algorithm requires empirical testing, and, as it is a naive approach to the problem, a faster yet more complicated solution may need to be implemented.
- We would like to examine the search speeds that result for a variety of tree structures. Currently, our trees contain about 5,000 observations. We would like to run tests on trees that contain all the observations in the IUE Minilog (about 70,000 records).
- To show the domain-independent nature of our approach, we will be constructing rule bases for other catalogs from a variety of different satellite observations.
- It would be useful if a library of successfully handled queries were maintained so that if a rule base is altered, the previously correctly handled queries can be rerun to ensure that a fix does not introduce a new error where there once was no problem. This utility exists in various expert system building tools, such as EMYCIN [5,18].
- We will likely be moving our interface from the Suntools environment to the more portable X-Windows system.

8 Summary and Conclusions

The spatial data management expert system is a large scale domain-independent system that serves as an intelligent front-end to databases containing spatial data. There are two major components to the system:

- The first is a spatial search module which uses the spatial component of the data in the database to produce a tree which contains the primary keys of all the records in the database. The spatial tree is indexed by the spatial part of a user's query, and a list is returned of all the primary keys that point to

records meeting that spatial criteria. We find that the time required to process a user's query is reduced dramatically, compared with the time needed by a DBMS that must necessarily search its entire database to discover which records satisfy the user's spatial demands.

- The second is a domain-independent query expert system (QES) that uses a domain-specific rule base to preprocess the user query, effectively mapping a broad class of queries into a smaller set that is manageable by a commercial natural language processing product. QES uses a forward-chaining inference engine that relies on the specificity of the rules and an indexing scheme to achieve a quiescent state rapidly, thus transforming the user's English query into a form that can be handled by DataTalker.

This system is a step toward automatically building intelligent user interfaces for the large, non-homogeneous databases that exist today and are being planned for the future. We feel we have shown that the techniques we have used can be incorporated into working application systems immediately. Systems which force users to use specialized database query languages to access any data should be rethought.

Acknowledgements

We would like to thank Dr. Michael Van Steenberg (National Research Council) for helping us with the IUE Minilog, providing us with sample user queries, and assuring us of the existence of a user base that needs this type of system. We are indebted to Craig Goettsche, of Science Applications Research (SAR), for tackling Suntools and implementing the region selection module and a help module. David Kortenkamp, currently a graduate student at the University of Michigan, helped design and implement some of the spatial search module during his summer internship with SAR. Finally, we would like to thank William Campbell (NASA/GSFC), Scott Wattawa (NASA/GSFC), Scott Hill (SAR), Nicholas Short, Jr. (NASA/GSFC) and Larry Roelofs (Computer Technology Associates) for their discussions and inputs into this research.

References

[1] Bentley, J. L., Multidimensional binary search trees used for associative searching, *Communications of the ACM,* **18** 9, 509-517, September 1975.

[2] Bentley, J. L., Programming pearls, *Communications of the ACM,* **28** 11, 1121-1127, 1985.

[3] Brownston, L., E. Kant, R. Farrell and N. Martin, *Programming Expert Systems in OPS5.* Reading, Massachusetts: Addison-Wesley, 1985.

[4] Campbell, W., N. Short, Jr., L. Roelofs and S. Wattawa, The intelligent user interface for NASA's advanced information management systems, *Third Conference on Artificial Intelligence for Space Applications, Part II*, 1987.

[5] Davis, R. and D. B. Lenat, *Knowledge-based Systems in Artificial Intelligence*, New York: McGraw Hill, 1982.

[6] Finkel, R. A., and Bentley, J. L. , Quad trees: a data structure for retrieval on composite keys, *Acta Informatica*, **4**, 1-9, 1974.

[7] Floyd, R. W. and Rivest, R. L., Expected time bounds for selection, *Communications of the ACM*, **18** 3, 165-172, 1975.

[8] Friedman, J. H., Bentley, J. L., and Finkel, R. A., An algorithm for finding best matches in logarithmic time, *Stanford CS Report*, 75-482.

[9] Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz and J. Slocum, Developing a natural language interface to complex data, *ACM Trans. on Database Systems*, **3** 2, 105-147, 1978.

[10] Knuth, D. E., *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley Publishing Company, 1973.

[11] Martin, P., D. Appelt and F. Pereira, Transportability and generality in a natural-language interface system, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 573-581, Los Altos: William Kaufmann, Inc., 1983.

[12] McDermott, J. and C. Forgy, Production system conflict resolution strategies, in D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, New York: Academic Press, 1978.

[13] Natural Language Incorporated, Natural language database retrieval system, version 3.0, Natural Language Incorporated, Berkeley, CA, 1988.

[14] Overmars, Mark H., and van Leeuwen, J., Dynamic multi-dimensional data structures based on quad- and k-d trees, *Acta Informatica*, **17**, 267-285, 1982.

[15] Short, N. Jr. and S. L. Wattawa, The second generation intelligent user interface for the crustal dynamics data information system, Telematics and Informatics, **5** 3, 253-268, 1988.

[16] Stroustrup, B., *The C++ Programming Language*, Addison-Wesley Publishing Company, 1986.

[17] Sybase Inc., Sybase database management, Sybase, Inc., Berkeley, CA, 1987.

[18] van Melle, W., A domain-independent system that aids in constructing knowledge-based consultation programs, Ph.D. dissertation, Stanford University, 1980.