Journal of Intelligent Information Systems, 6, 1?? (1996)

CoBase: A Scalable and Extensible Cooperative Information System*

WESLEY W. CHU, HUA YANG, KUORONG CHIANG,
MICHAEL MINOCK, GLADYS CHOW, CHRIS LARSON

{wwc, hua}@cs.ucla.edu

Computer Science Department, University of California, Los Angeles, CA 90095

Editor: Gio Wiederhold

Abstract. A new generation of information systems that integrates knowledge base technology with database systems is presented for providing cooperative (approximate, conceptual, and associative) query answering. Based on the database schema and application characteristics, data are organized into Type Abstraction Hierarchies (TAHs). The higher levels of the hierarchy provide a more abstract data representation than the lower levels. Generalization (moving up in the hierarchy), specialization (moving down the hierarchy), and association (moving between hierarchies) are the three key operations in deriving cooperative query answers for the user. Based on the context, the TAHs can be constructed automatically from databases. An intelligent dictionary/directory in the system lists the location and characteristics (*e.g.*, context and user type) of the TAHs. CoBase also has a relaxation manager to provide control for query relaxations. In addition, an explanation system is included to describe the relaxation and association processes and to provide the quality of the relaxed answers. CoBase uses a mediator architecture to provide scalability and extensibility. Each cooperative module, such as relaxation, association, explanation, and TAH management, is implemented as a mediator. Further, an intelligent directory mediator is provided to direct mediator requests to the appropriate service mediators. Mediators communicate with each other via KQML. The GUI includes a map server which allows users to specify queries graphically and incrementally on the map, greatly improving querying capabilities. CoBase has been demonstrated to answer imprecise queries for transportation planning. Currently, we are applying the CoBase methodology to match medical image (X-ray, MRI) features and approximate matching of emitter signals in electronic warfare applications.

Keywords: Approximate query answering, query relaxation, associative query answering, explanation system, mediator architecture, type abstraction hierarchy, conceptual clustering

1. Introduction

Consider asking a query of a human expert. If the posed query has no answer or the complete data for an answer is not available, you do not simply get a null response. The human expert attempts to understand the gist of your query, to suggest or answer related questions, to infer an answer from data that is accessible, or to give an approximate answer. The goal of cooperative database research is to create information systems with these characteristics [14]. The key is the integration of a knowledge base which represents the data semantics.

* This work supported by ARPA contract F30602-94-C-0207.

In conventional databases, if required data is missing, if an exact answer is unavailable, or if a query is not well-formed with respect to the schema, the database just returns a null answer or an error. An intelligent system would be much more resourceful and cooperative, permitting conceptual level queries (containing concepts that may not be expressed in the database schema) when the user does not know the precise schema, providing approximate answers when some data is missing, or even volunteering associative (relevant) information to the query answer. Cuppens and Demolombe [6] provide cooperative answers by rewriting the query to add variables to the query vector, which carry relevant information to the user. The rewrite is not a generalization of the query, it is an extension of the query to provide more information. Motro [22] proposes allowing the user to select the direction of relaxation and thus to indicate which relaxed answers may be of interest. Hemerly, *et al.* [16] uses a predefined user model and maintains a log of previous interactions to avoid misconstruction when providing additional information. All the above approaches are rule-based and difficult to scale up. To remedy this shortcoming, we will present a structured approach and its implementation to cooperative query answering for database systems.

In this paper, we shall first present the concept of Type Abstraction Hierarchy (TAH) [4], [3] to provide a structured approach for query modification. Methodologies for automatic TAH generation are discussed. Next, we present the cooperative SQL primitives and selected examples. Then, we present the scalable cooperative information system: the relaxation controls for providing efficient query processing and filtering out unsuitable answers for the user, a TAH mediator for providing management and editing of the TAHs in the system, the case-based approach for providing association information to query answers, an explanation mediator, and the inter-mediator communications. Implementation issues are also discussed. The performance of CoBase from a set of sample queries generated from the testbed is reported. Finally, we discuss the technology transfer of CoBase to transportation, medical image databases, and electronic warfare applications.

2. Structured Approach to Cooperative Query Answering

2.1. Query Relaxation via Type Abstraction Hierarchies

Cooperative query answering relaxes a query scope to enlarge the search range or relaxes an answer scope to include additional information. Enlarging and shrinking a query scope can be accomplished by viewing the queried objects at different conceptual levels, since an object representation has wider coverage at a higher level and inversely, more narrow coverage at a lower level. Although linking different level object representations can be made in terms of explicit rules [6], such linking lacks a systematic organization to guide the query transformation process. To remedy this problem, we propose the notion of a type abstraction hierarchy (TAH) [4], [2] for providing an efficient and organized framework for cooperative query processing. A TAH represents objects at different levels of abstraction. For

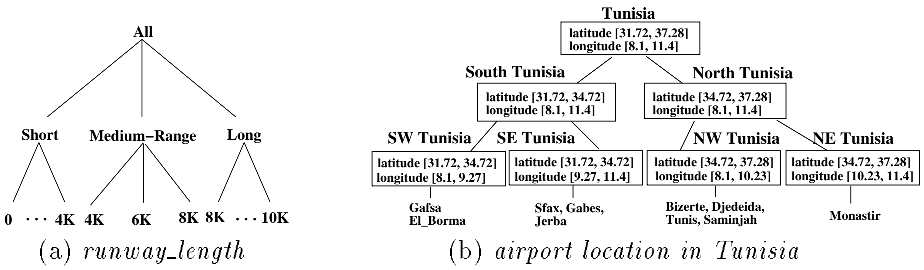

Figure 1. Type Abstraction Hierarchies

example, in Figure 1, the *Medium-Range* (*i.e.*, from 4,000 to 8,000 ft.) in the TAH for *runway-length* is a more abstract representation than a specific runway length in the same TAH (*e.g.*, 6,000 ft). Likewise, *SW Tunisia* is a more abstract representation than individual airports (*e.g.*, *Gafsa*). A higher-level and more abstract object representation corresponds to multiple lower-levels and more specialized object representations. Querying an abstractly represented object is equivalent to querying multiple specialized objects.

A query can be modified by relaxing the query conditions via such operations as *generalization* (moving up the TAH) and *specialization* (moving down the TAH), e.g., from 6000ft to Medium-Range to [4000ft, 8000ft]. In addition, queries may have *conceptual* conditions such as "runway-length = Medium-Range." This condition can be transformed into *specific* query conditions by specialization. Query modification may also be specified explicitly by the user through a set of *cooperative operators* such as 'similar-to', 'approximate', 'near-to', *etc.* This approach was also adopted in the cooperative deductive databases by Gaasterland, *et al.* [15] for providing query relaxations.

The notion of multi-level object representation is not captured by the conventional semantic network and object-oriented database approaches for the following reasons: grouping objects into a class and grouping several classes into a super-class only provide a common "title" (type) for the involved objects without concern for the object instance values and without introducing abstract object representations. Grouping several objects together and identifying their aggregation as a single (complex) object does not provide abstract instance representations for its component objects. Therefore, an object-oriented database deals with information only at two general layers: the meta-layer and the instance layer. Since forming an object-oriented type hierarchy does not introduce new instance values, it is impossible to introduce an additional instance layer. In the Type Abstraction Hierarchy, instances of a super-type and a sub-type may have different representations and can be viewed at different instance layers. Such multiple layer knowledge representation is essential for cooperative query answering.

Knowledge for query relaxation can be expressed as a set of logical rules, but such a rule-based approach [7], [16] lacks a systematic organization to guide the query transformation process. TAHs provides a much simpler and more intuitive representation for query relaxation, and do not have the complexity of the inference that exists in the rule-based system. As a result, the TAH structure can easily support flexible relaxation control (see Section 5.2.1), which is important to improve relaxation accuracy and efficiency. Further, knowledge represented in a TAH is customized, thus changes in one TAH represent only a localized update and do not affect other TAHs, simplifying TAH maintenance (see Section 3.3). We have developed tools to generate TAHs automatically from data sources (see Section 3), which enable our system to scale up and extend to large data sources.

2.2. Properties of the Type Abstraction Hierarchy

Type Abstraction Hierarchy (TAH) is a representation for the abstraction of individual types (atomic or tuple) at instance levels, which is different from the meta-level class/type hierarchy in object-oriented systems.

In our notion of type abstraction, a type can be atomic T (for single attribute), or tuple $T : (T_1, \ldots, T_m)$ (for multiple attributes), where $T_1, ..., T_m$ are atomic types. Let us first define the concept of type abstraction.

Type T' is an abstraction of type T, denoted as $T \preceq T'$, means the following:

1. For atomic types T and T', each value in $dom(T')$ represents single or multiple values in $dom(T)$, where $dom(T)$ denotes the domain of type T and $dom(T')$ is the domain of type T'.
2. For tuple types $T : (T_1, \ldots, T_m)$ and $T' : (T'_1, \ldots, T'_m)$, $\forall\, i \in \{1, \cdots, m\}, T_i \preceq T'_i$.

Typically, the abstraction of a numerical type is a range (or interval) type while the abstraction of a nonnumerical type is a type of conceptual names (*e.g.*, *SW Tunisia*), each representing a set of values (*e.g.*, *Gafsa, El Borma*). We can also assign conceptual names to ranges (*e.g.*, *long runway*).

A TAH is a tree structure constructed from a set of values of a ***base type***, which is the most specific type. All other types are abstraction of the base type or other abstract types. The values of the base type are leaf nodes of the TAH. Conceptually, all leaf nodes have the same depth in the TAH[1]. The values of the abstract types constitute the non-leaf nodes of the TAH.

More formally, we denote the base type as T and all abstract types in the TAH as T', T'', $\ldots$, $T^{(k)}$, where $T^{(k)}$ is the most general type. We have

$$T \preceq T' \preceq T'' \preceq \ldots \preceq T^{(k)}$$

For convenience of notation, T, T', T'', $\ldots$ can also be re-written as $T^{(0)}$, $T^{(1)}$, $T^{(2)}$, $\ldots$, respectively. Values of type $T^{(i)}$ are at i levels above the leaf nodes for $i = 1, \ldots, k$. If we construct a TAH based on a single attribute, then the attribute is the base type. If we construct a TAH based on multiple attributes chosen from one or more relations, then these attributes constitute the base tuple type.

Intuitively, moving up a TAH maps a value of type $T^{(i)}$ to a value of more abstract type $T^{(i+1)}$, which is more general; while moving down the TAH maps from a value of more abstract type $T^{(i+1)}$ to a set of values of type $T^{(i)}$, which is more specific, where $i = 0, \ldots, k-1$. Relaxation for a value (of the base type) is to map it to a more abstract value and then map this value back to a set of base type values, thus covering a larger scope.

Let $x^{(i)}$ be a value of type $T^{(i)}$, where $i = 0, 1, \ldots, k$. Since each value corresponds to a node in the TAHs, moving from one node $x^{(i)}$ up one level to another node $x^{(i+1)}$ in a TAH can be accomplished by the function $generalize(x^{(i)})$ and moving from a node $x^{(i)}$ down to the bottom $x^{(0)}$ can be accomplished by $specialize(x^{(i)})$. The result of $generalize(x^{(i)})$ is always a single value and of a type more abstract than the type of $x^{(i)}$ (*i.e.*, some $x^{(i+1)}$), and values in $specialize(x^{(i)})$ will be the leaf nodes (*i.e.*, the value of the base type $T^{(0)}$) as defined recursively,

$$specialize(x^{(i)}) = \begin{cases} \{x^{(0)}\} & \text{if } i = 0 \ (i.e., \text{ the leaf node}) \\ \bigcup_{\forall x^{(i-1)}} specialize(x^{(i-1)}) & \text{otherwise} \end{cases}$$

where $x^{(i-1)} \in \{x^{(i-1)} | x^{(i)} = generalize(x^{(i-1)})\}$.

Then a single step relaxation for a value $x^{(i)}$ is defined as

$$relax(x^{(i)}) = specialize(generalize(x^{(i)}))$$

For a value $x^{(i)}$ of type $T^{(i)}$, it can be relaxed $k - i$ times at most. A j-step relaxation (where $j = 1, 2, \ldots, k$) can be obtained by applying $generalize()$ j times and then applying $specialize()$ once to the result:

$$relax^{(j)}(x^{(i)}) = specialize(generalize^{j}(x^{(i)})), \ where \ i \leq k - j$$

Relaxation has the following property:

$$relax^{(j)}(x^{(i)}) \subseteq relax^{(j+1)}(x^{(i)}), for \ i = 0, \ldots, k-2 \ and \ j = 1, \ldots, k-i-1.$$

The above definition is also applicable to cases where the base types are tuple types (*i.e.*, MTAHs).

3. Automatic Knowledge Acquisition

The automatic generation of knowledge base (TAHs) from databases is essential for CoBase to be scalable to large systems. We have developed algorithms to automatically generate TAHs based on database instances. A brief discussion about the algorithms and their complexity are followed.

3.1. Numerical TAHs

COBWEB [10], a conceptual clustering system, uses *category utility* (CU) [13] as a quality measure to classify the objects described by a set of attributes into a classification tree. Formally, for a partition from a class C to N mutually exclusive classes $C_1, ..., C_N$, the category utility (CU) is defined as the increase in the *goodness*[2] of these classes after partition. That is,

$$CU(C_1, ..., C_N) = \frac{\sum_{k=1}^{N} P(C_k)G(C_k) - G(C)}{N} \tag{1}$$

where $P(C_k)$ is the occurrence probability of C_k in C, and $G(C_k)$ and $G(C)$ are the goodness functions for C_k and C, respectively. That is,

$$G(C_k) = \sum_{a \in A} \sum_{x_i^a \in X_k^a} P(x_i^a)^2 \tag{2}$$

$$G(C) = \sum_{a \in A} \sum_{x_i^a \in X^a} P(x_i^a)^2 \tag{3}$$

where A is the set of all the attributes, and X_k^a and X^a are the distinct values of attribute a in C_k and C respectively.

COBWEB cannot be used for abstracting numerical data; it only deals with categorical data. Moreover, its classification tree serves as *the database* for instances and requires a large storage space. Furthermore, matching of objects with existing classes is time consuming. For providing approximate answers, we want to build a classification tree that minimizes the difference between the desired answer and the derived answer. Specifically, we use relaxation error as a measure for clustering. The *relaxation error* (RE) is defined as the average difference between the requested values and the returned values. $RE_1(C)$ can also be interpreted from the standpoint of query relaxation. Let us define the *relaxation error of* x_i, $RE_1(x_i)$, as the average difference from x_i to x_j, j=1,...,n, that is,

$$RE_1(x_i) = \sum_{j=1}^{n} P(x_j)|x_i - x_j| \tag{4}$$

where $P(x_j)$ is the occurrance probability of x_j in C. $RE_1(x_i)$ can be used to measure the quality of an approximate answer where x_i in a query is relaxed to x_j, j=1,...,n. Summing $RE_1(x_i)$ over all values x_i in C, we have

$$RE_1(C) = \sum_{i=1}^{n} P(x_i)RE_1(x_i). \tag{5}$$

Thus, $RE_1(C)$ is the expected error of relaxing any value in C.

If $RE_1(C)$ is large, query relaxation based on C may produce very poor approximate answers. To overcome this problem, we can partition C into sub-clusters to reduce relaxation error. Given a partition $P = \{C_1, C_2, ..., C_N\}$ of C, the *relaxation error of the partition* P is defined as

$$RE_1(P) = \sum_{k=1}^{N} P(C_k) RE_1(C_k) \tag{6}$$

where $P(C_k)$ equals the number of tuples in C_k devided by the number of tuples in C. In general, $RE_1(P) < RE_1(C)$.

Relaxation error is the expected pair-wise difference between values in a cluster. To extend the notion of relaxation error from a single attribute to multiple attributes, we shall consider distance between tuples instead of difference between values. Given two m-attribute tuples $t_i = \{x_1, ..., x_m\}$ and $t_j = \{y_1, ..., y_m\}$, their distance is defined as

$$D(t_i, t_j) = \sum_{k=1}^{m} W_k \frac{|x_k - y_k|}{\Delta_k} \tag{7}$$

where W_k and Δ_k are the weight and the normalization constant for the k-th attribute, respectively. W_k can be assigned to reflect relative importance among attributes. Δ_k is necessary for summing up differences from different attributes because these attributes have different distributions.

Given a cluster of n m-attribute tuples $C = \{t_1, ..., t_n\}$, the relaxation error for C is defined as the average pair-wise distance among tuples in C, that is,

$$RE(C) = \frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{n} P(t_i) P(t_j) D(t_i, t_j) \tag{8}$$

where $P(t_i)$ and $P(t_j)$ are the probabilities of tuples t_i and t_j, respectively. The division by m in (8) normalizes RE(C) per attribute and allows us to compare relaxation errors computed from different numbers of attributes. The category utility (CU) for multiple attributes can be obtained by simply substituting $RE_1(C)$ in Eq (9) by its multi-attribute counterpart $RE(C)$ in Eq (8).

Using relaxation error, the category utility can be defined as the relaxation error reduction per sub-cluster, that is,

$$\begin{aligned} CU &= \frac{\sum_{k=1}^{N} P(C_k)[1 - RE_1(C_k)] - [1 - RE_1(C)]}{N} \\ &= \frac{RE_1(C) - \sum_{k=1}^{N} P(C_k) RE_1(C_k)}{N}. \end{aligned} \tag{9}$$

DIstribution Sensitive Clustering (DISC) [3] partitions sets of numerical values into clusters that minimize the relaxation error. We shall now present a class of DISC algorithms for clustering numerical values. We shall present the algorithm for a single attribute, and then extend it for multiple attributes.

The Clustering Algorithm for a Single Attribute

Given a cluster with n distinct values, the number of partitions is exponential with respect to n, so the best partition according to (9) takes exponential time to find.

Algorithm DISC(C)
  if the number of distinct values $\in C < T$, return /* T is a threshold */
  let cut = the best cut returned by **BinaryCut**(C)
  partition values in C based on cut
  let the resultant sub-clusters be C_1 and C_2
    call DISC(C_1) and DISC(C_2)

Algorithm BinaryCut(C)
/* input cluster $C = \{x_1, ..., x_n\}$ */
  for $h = 1$ to $n - 1$ /* evaluate each cut */
    Let P be the partition with clusters $C_1 = \{x_1, ..., x_h\}$ and $C_2 = \{x_{h+1}, ..., x_n\}$
    compute $RE_1(P)$
    if $RE_1(P) < MinRE$ then
      $MinRE = RE_1(P)$, $cut = h$ /* the best cut */
  Return cut as the best cut

Table 1. The algorithms DISC and BinaryCut

To reduce computation complexity, we shall only consider binary partitions (i.e., $N = 2$ in (9)). Later we shall show a simple hill climbing strategy can be used for obtaining N-ary partitions from binary partitions.

Our method is top down: we start from one cluster consisting of all the values of an attribute, and then we find "cuts"[3] to recursively partition the cluster into smaller clusters. The partition result is a concept hierarchy called Type Abstraction Hierarchy (TAH). The clustering algorithm is called the DISC (DIstribution Sensitive Clustering) Method and is given in Table 1.

In [3], an implementation of the algorithm BinaryCut is presented whose time complexity is $O(n)$. Since DISC needs to execute BinaryCut $n-1$ times at most to generate a TAH, the worst case time complexity of DISC is $O(n^2)$. (The average case time complexity of DISC is $O(n \log n)$.)

N-ary Partitioning

N-ary partitions can be obtained from binary partitions by a hill climbing method. Starting from a binary partition, the sub-cluster with greater relaxation error is selected for further cutting. We shall use CU as a measure to determine if the newly formed partition is better than the previous one. If the CU of the binary partition is greater than that of the tri-nary partition, then the tri-nary partition is dropped and the cutting is terminated. Otherwise, the tri-nary partition is selected and the cutting process continues until it reaches the point where a cut decreases CU. The procedure is outlined in Table 2.

The Clustering Algorithm for Multiple Attributes

Query relaxation for multiple attributes using multiple single-attribute TAHs relaxes each attribute independently disregarding the relationships that might exist

Algorithm N-ary Partition(C)
let C_1 and C_2 be the two sub-clusters of C
compute CU for the partition C_1, C_2
for $N = 2$ to $n - 1$
    let C_i be the sub-cluster of C with maximum relaxation error
    call BinaryCut to find the best sub-clusters C_{i1} and C_{i2} of C_i
    compute and store CU for the partition $C_1, ..., C_{i-1}, C_{i1}, C_{i2}, C_{i+1}, ..., C_N$
    if current CU is less than the previous CU
        stop
    else
        replace C_i by C_{i1} and C_{i2}
/* the result is an N-ary partition of C */

Table 2. The N-ary partition algorithm

among attributes. This may not be adequate for the applications where attributes are dependent.[4] In addition, using multiple single attribute TAHs is inefficient since it may need many iterations of query modification and database access before approximate answers are found. Furthermore, relaxation control for multiple TAHs is more complex since there is a large number of possible orders for relaxing attributes. In general, we can only rely on simple heuristics such as *best first* or *minimal coverage first* to guide the relaxation (see Section 5.2.1). These heuristics cannot guarantee best approximate answers since they are rules of thumb and not necessarily accurate.

Most of the above mentioned difficulties can be overcome by using Multi-attribute TAH (MTAH) for the relaxation of multiple attributes. Since MTAHs are generated from *semantically dependent* attributes, these attributes are relaxed together in a single relaxation step, thus greatly reducing the number of query modification and database access. Approximate answers derived by using MTAH have better quality than those derived by using multiple single-attribute TAHs. MTAHs are context and user sensitive because a user may generate several MTAHs with different attribute sets from a table. Should a user needs to create an MTAH containing semantically-dependent attributes from different tables, these tables can be joined into a single view for MTAH generation.

To cluster objects of multiple attributes, DISC can be extended to M-DISC (shown in Table 3). The generated multi-dimensional TAHs are called MTAHs. The algorithm DISC is a special case of M-DISC, and TAH is a special case of MTAH. Let us now consider the time complexity of M-DISC. Let m be the number of attributes and n be the number of distinct attribute values. The computation of relaxation error for a single attribute takes $O(n \log n)$ to complete [3]. Since the computation of CU involves computation of relaxation error for m attributes, its complexity is $O(mn \log n)$. The nested loop in M-DISC is executed mn times, so the time complexity of M-DISC is $O(m^2 n^2 \log n)$. To generate an MTAH, it takes no more than n calls of M-DISC, therefore, the worst case time complexity

```
Algorithm M-DISC(C)
    if the number of objects in C < T, return /* T is a threshold */
    for each attribute a = 1 to m
        for each possible binary cut h
            compute CU for h
            if CU > MaxCU then /* remember the best cut */
                MaxCU = CU, BestAttribute = a, cut = h
    partition C based on cut of the attribute BestAttribute
    let the resultant sub-clusters be C1 and C2
        call M-DISC(C1) and M-DISC(C2)
```

Table 3. Multi-attribute DISC (M-DISC) algorithm

of generating an MTAH is $O(m^2 n^3 \log n)$. The average case time complexity is $O(m^2 n^2 (\log n)^2)$ since M-DISC needs only to be called $\log n$ times on the average.

3.2. Non-numerical TAHs

For generating TAHs with non-numerical attributes, the values are clustered based on inter-attribute relationships. A Pattern-Based Knowledge Induction (PBKI) algorithm [19] is developed to compute the pair-wise correlations among attribute values. Correlation is then used as a nearness measure in clustering the attribute values. A more detailed discussion about the algorithm is presented in [19].

3.3. Maintenance of TAHs

Since the quality of TAH affects the quality of derived approximate answers, TAHs should be kept up to date. One simple way for maintaining TAHs is to regenerate them whenever an update occurs. This approach is not desirable because it causes overhead for the database system. Although each update changes the distribution of data (thus changing the quality of the corresponding TAHs), this may not be significant enough to warrant a TAH regeneration. TAH regeneration is only necessary when the cumulative effect of updates has greatly degraded the TAHs. The quality of a TAH can be monitored by comparing the derived approximate answers to the expected relaxation error (for example, see Figure 8) which is computed at TAH generation time and recorded at each node of the TAH. When the derived approximate answers significantly deviate from the expected quality, then the quality of the TAH is deemed to be inadequate and a regeneration is necessary. The following incremental TAH regeneration procedure can be used. First, identify the node within the TAH that has the worst query relaxations. Apply partial TAH regeneration for all the database instances covered by the node. After several such partial regenerations, we then initiate a complete TAH regeneration.

The generated TAHs are stored in Unix files and a TAH Manager (described in Section 5.3) is responsible to parse the files, create internal representation of TAHs, and provide operations such as generalization and specialization to traverse TAHs. The TAH Manager also provides a directory that describes the characteristics of TAHs (*e.g.*, attributes, names, user type, context, TAH size, location) for the users/systems to select the appropriate TAH to be used for relaxation.

Our experience in using DISC/MDISC and PBKI for ARPI transportation databases (94 relations, the biggest one of which has 12 attributes and 195,598 tuples) shows that the clustering techniques for both numerical and non-numerical attributes can be generated from a few seconds to a few minutes depending on the table size on a Sun SPARC 20 Workstation.

4. Cooperative SQL (CoSQL)

4.1. CoSQL Cooperative Operators

The cooperative operations consist of the following four types: context free, context sensitive, control, and interactive. The context free and context sensitive cooperative operators can be used in conjunction with attribute values specified in the WHERE clause. The relaxation control operators can only be used on attributes specified in the WHERE clause, and the control operators have to be specified in the WITH clause after the WHERE clause. The interactive operators can be used alone as command inputs.

Context free operations.

- *Approximate operator*, $\wedge v$, relaxes the specified value v within the approximate range that is predefined by the user. For example, $\wedge$9am transforms into the interval (8am, 10am).
- *Between* (v_1, v_2) specifies the interval for an attribute. For example, time between (7am, $\wedge$9am) transforms into (7am, 10am). The transformed interval is pre-specified either by the user or the system.
- *Within* $(x_1, y_1, ..., x_n, y_n)$ specifies a region on a map by a closed polygon $(x_1, y_1, ..., x_n, y_n)$.

Context sensitive.

- *Near-to* X is used for specification of spatial nearness of object X. The "near-to" measure is context and user sensitive. "Nearness" can be determined by the TAHs or specified by the user. For example, `near-to 'BIZERTE'` requests the list of cities in the same cluster as `BIZERTE` in a TAH, or cities located within a certain Euclidean distance (depending on the context) from the city 'BIZERTE'.
- *Similar-to* X *based-on* $((a_1\ w_1)(a_2\ w_2)...(a_n\ w_n))$ is used to specify a set of objects semantically similar to the target object X based on a set of attributes

$(a_1, a_2, ..., a_n)$ specified by the user. Weights $(w_1, w_2, ..., w_n)$ may be assigned to each of the attributes to reflect the relative importance in considering the similarity measure. The set of similar objects can be ranked by the similarity. The similarity measures that computed from the nearness (e.g. weighted mean square error) of the pre-specified attributes to that of the target object. The set size is bound by a pre-specified nearness threshold.

Control Operators.

- *Relaxation-order* $(a_1, a_2, ..., a_n)$ specifies the order of the relaxation among the attributes $(a_1, a_2, ..., a_n)$ (*i.e.*, a_i precedes a_{i+1}). For example, `relaxation-order (runway_length, runway_width)` indicates that if no exact answer is found, then runway_length should be relaxed first. If still no answer is found, then relax the runway_width. If no relaxation-order control is specified, the system relaxes according to its default relaxation strategy.
- *Not-relaxable* $(a_1, a_2, ..., a_n)$ specifies the attributes $(a_1, a_2, ..., a_n)$ that should not be relaxed. For example, `not-relaxable location_name` indicates that the condition clause containing location_name must not be relaxed.
- *Preference-list* $(v_1, v_2, ..., v_n)$ specifies the preferred values $(v_1, v_2, ..., v_n)$ of a given attribute, where v_i is preferred over v_{i+1}. As a result, the given attribute is relaxed according to the order of preference that the user specifies in the preference list. For example, the attribute "food style," a user may prefer Italian food to Mexican food. If there are no such restaurants within the specified area, the query can be relaxed to include the foods similar to Italian food first, and then similar to Mexican food.
- *Unacceptable-list* $(v_1, v_2, ..., v_n)$ allows users to inform the system not to provide certain answers. This control can be accomplished by trimming parts of the TAH from searching. For example, "avoid airlines X and Y" tells the system that airlines X and Y should not be considered during relaxation. It not only provides more satisfactory answers to users, but also reduces search time.
- *Alternative-TAH (TAH-name)* allows users to use the TAHs of their choices. For example, a vacation traveler may want to find an airline based on its fare while a business traveler is more concerned with his schedule. To satisfy the different needs of the users, several TAHs of airlines can be generated, emphasizing different attributes (*e.g.*, price and nonstop flight).
- *Relaxation-level* (v) specifies the maximum allowable range of the relaxation on an attribute, *i.e.*, $[0, v]$.
- *Answer-set* (s) specifies the minimum number of answers required by the user. CoBase relaxes query conditions until enough number of approximate answers (*i.e.*, $\geq s$) are obtained.

User/System interaction operators.

- *Nearer, Further* provide users with the ability to control the "near-to" relaxation scope interactively. Nearer reduces the distance by a pre-specified percentage while further increases the distance by a pre-specified percentage.

4.2. Editing Relaxation Control Parameters

Users can browse and edit relaxation control parameters to better suit their applications (for example, Figure 2). The parameters include the relaxation range for the "approximately-equal" operator, the default distance for the "near-to" operator, the number of returned tuples for the "similar-to" operator, *etc.*

4.3. Examples

In this section, we present a few selected examples that illustrate the capabilities of the cooperative operators. The corresponding TAHs used for query modification are shown in Figure 1 and the default relaxable ranges are shown in Figure 2.
Query 1. List all the airports with the runway length greater than 7500 feet and runway width greater than 100 feet. If there is no answer, relax the runway length condition first. The following is the corresponding CoSQL query:

```
SELECT aport_name, runway_length_ft, runway_width_ft
FROM aports
WHERE runway_length_ft > 7500 AND runway_width_ft > 100
WITH  RELAXATION-ORDER (runway_length_ft, runway_width_ft)
```

Based on the runway-length TAH and the relaxation order, the query is relaxed to:

```
SELECT aport_name, runway_length_ft, runway_width_ft
FROM aports
WHERE runway_length_ft >= 4000 AND runway_width_ft > 100
```

If this query yields no answer, then we proceed to relax the range of runway width by a pre-specified value (*i.e.*, 10 ft) as shown in Figure 2.
Query 2. Find all the airports with their geographical coordinates near Bizerte in the country Tunisia. If there is no answer, the restriction on the country should not be relaxed. The corresponding CoSQL query is as follows:

```
SELECT aport_name, latitude, longitude
FROM aports, GEOLOC
WHERE aport_name NEAR-TO 'Bizerte'
      AND country_state_name = 'Tunisia'
      AND GEOLOC.geo_code = aports.geo_code
WITH  NOT-RELAXABLE country_state_name
```

Based on the TAH on airport location in Tunisia (Figure 1), the relaxed query is:

Approximate Operator Relaxation Range

Relation Name	Attribute Name	∧Range
aports	runway_width_ft	10
aports	parking_sq_ft	100000

Near-to Operator Relaxation Range

Relation Name	Attribute Name	Near-to Range	Near/Further
aports	aport_name	100 miles	50%

Figure 2. Relaxation Range for the Approximate and Near-to Operators.

```
SELECT aport_name, latitude, longitude
FROM aports, GEOLOC
WHERE aport_name IN {'Bizerte', 'Djedeida',
                     'Saminjah', 'Tunis'}
      AND GEOLOC.geo_code = aports.geo_code
```

If the TAH for the airport location in Tunisia is not available, a pre-defined range (Figure 2) can be used, which relaxes the search range by 100 miles from Bizerte.

Query 3. Find all airports in Tunisia similar to the Bizerte airport. Use the attributes runway_length_ft, runway_width_ft as criteria for similarity. Place more similarity emphasis on runway length than runway width; their corresponding weight assignments are 2 and 1, respectively. The following is the CoSQL version of the query:

```
SELECT aport_name
FROM aports, GEOLOC
WHERE aport_name SIMILAR-TO 'Bizerte'
                 BASED-ON ((runway_length_ft 2.0)
                           (runway_width_ft 1.0))
      AND country_state_name = 'TUNISIA'
      AND GEOLOC.geo_code = aports.geo_code
```

To select the set of airport names that have the runway length and runway width similar to those of the Bizerte airport, we shall first find the values of the attributes: runway_length_ft and runway_width_ft for the Bizerte airport, therefore, transforming the query to:

```
SELECT runway_length_ft, runway_width_ft
FROM aports, GEOLOC
WHERE country_state_name_ = 'TUNISIA'
      AND GEOLOC.geo_code = aports.geo_code
      AND aport_name = 'Bizerte'
```

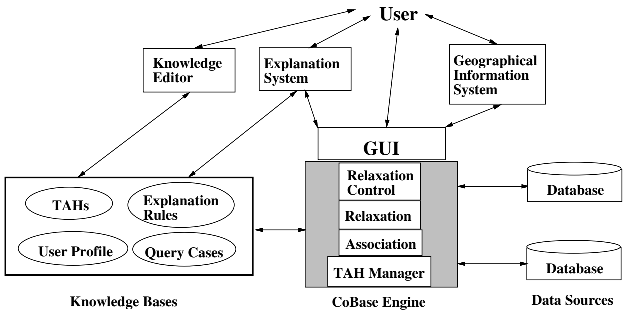

Figure 3. CoBase Functional Architecture.

The system relaxes the values of the attributes: runway_length_ft and runway_width_ft using an MTAH or TAHs until the size of the similarity set is satisfied by the pre-specified answer set size. The system then computes the similarity of these airports to "Bizerte" using the pre-specified *nearness* formula (*e.g.*, weighted mean squared error). The order in the similarity set is ranked according to the nearness measure.

5. A Scalable and Extensible Architecture

Figure 3 shows an overview of the CoBase System. Type abstraction hierarchies and relaxation ranges for the explicit operators are stored in a knowledge base (KB). There is a TAH directory storing the characteristics of all the TAHs in the system. When CoBase asks queries, it asks the underlying database systems (DBMS). When an approximate answer is returned, the user can ask for an explanation of how the answer was derived or an annotated relaxation path. A context-based semantic nearness will be provided to rank the approximate answers (in order of nearness) against the specified query. A GUI displays the query, results, TAHs, and explanations of the relaxation processes. Based on user type and query context, associative information is derived from past query cases. A user can construct TAHs from one or more attributes and modify the existing TAH in the KB.

5.1. Mediator Architecture

We use the concept of *mediation* [26] to decompose our cooperative answering system into reusable and interacting components. A *Mediator* is a software module that takes some input set of information, intelligently analyzes the information from a specific viewpoint, and produces a set of conclusions based on its analysis. Often, a Mediator needs additional information (knowledge and/or data) to perform its analysis. This information can be required as inputs to the Mediator's analysis, the Mediator can seek the assistance of other Mediators in fulfilling its information

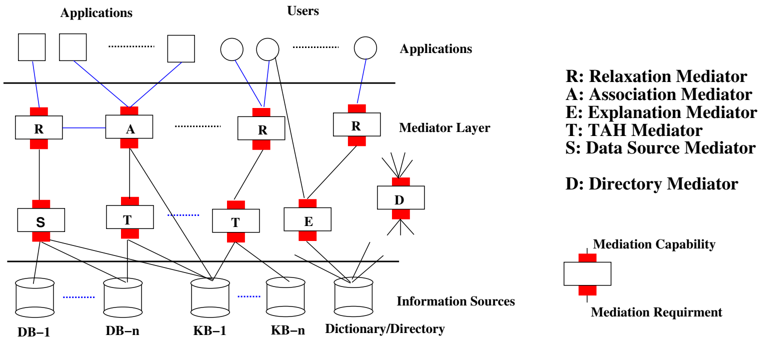

Figure 4. A Scalable and Extensible Cooperative Information System

needs. This latter mode (Mediators assisting Mediators) is called *dynamic matching*. Specifically, a Mediator with an information need will report this need to the environment and dynamically match with the Mediators that can fulfill the need.

Our use of Mediators is to decompose the cooperative query answering capabilities, where:

- Each mediator is composed of (1) its mediation process; (2) its mediation capabilities specifying what the mediator produces given a specific input set; and (3) its mediation requirements specifying what information the mediator needs access during processing (information that is not the postcondition input).

- One mediator's requirements can match (or link to) another mediator's capabilities – this represents the linking mediator's matching with the linked to mediator's information process.

- One or more *Directory Mediators* are needed to act as the information repository of the mediator set. Specifically, the directory mediator catalogs the mediators available in the system by their capabilities and requirements. It can then be consulted to find a mediator which meets a specific information need.

Figure 4 displays the various cooperative mediators: Relaxation, TAH , Association, Explanation, Data Source, and Directory. These mediators are connected selectively to meet applications' needs. An application that requires relaxation and explanation capabilities, for example, will entail a linking of Relaxation and Explanation mediators. Our mediator architecture allows incremental growth with application. When the demand for certain mediators increases, additional copies of the mediators can be added to reduce the loading, thus the system is scalable. For example, there are multiple copies of relaxation mediator and TAH mediator in Figure 4. Further, different types of mediators can be interconnect together and communicate with each other via a common communication protocol (*e.g.*, KQML) to perform a joint task. Thus, the architecture is extensible.

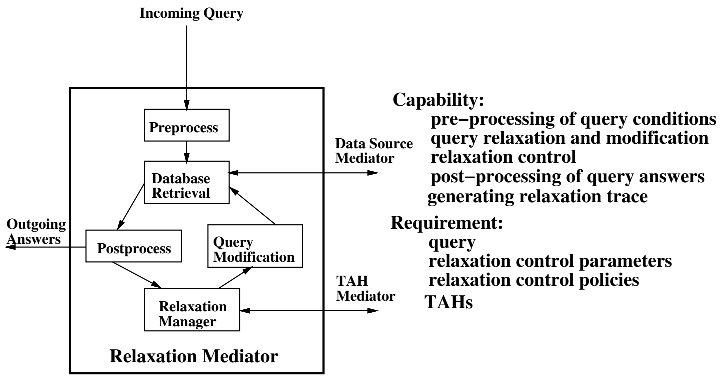

Figure 5. The Relaxation Mediator

5.2. Relaxation Mediator

Figure 5 illustrates the functional components of the relaxation mediator, its capability, and input requirements. Query relaxation is the process of understanding the semantic context, intent of a user query and modifying the query constraints with the guidance of the customized knowledge structure (TAH) into "near" values that provide "best-fit" answers. The flow of the relaxation process is depicted in Figure 6. When a CoSQL query is presented to the Relaxation Mediator, the system first go through a pre-processing phase. During the pre-processing, the system first relaxes any context free and/or context sensitive cooperative operators in the query. All relaxation control operations specified in the query will be processed. The information will be stored in the relaxation manager and ready to be used if the query requires relaxation. The modified SQL query is then presented to the underlying database system for execution. If no answers are returned, then the cooperative query system, under the direction of the Relaxation Manager, relaxes the queries by query modification. This is accomplished by traversing along the TAH node for performing generalization and specialization and rewriting the query to include a larger search scope. The relaxed query is then executed, and if there is no answer, we repeat the relaxation process until we obtain one or more approximate answers. If the system fails to produce an answer due to over-trimmed TAHs, the relaxation manager will deactivate certain relaxation rules to restore part of a trimmed TAH to broaden the search scope until answers are found. Finally, the answers are post-processed (*e.g.*, ranking, filtering, *etc*).

5.2.1. Relaxation Control

Relaxation without control may generate more approximations than the user can handle. The policy for relaxation control depends on many factors, including user

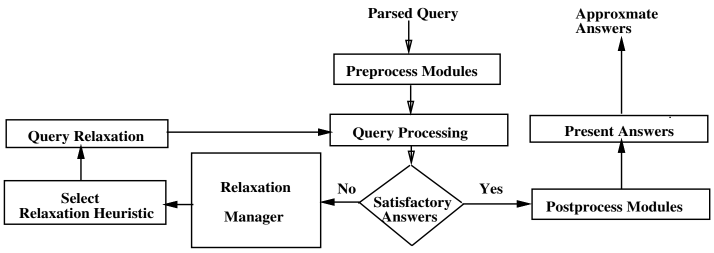

Figure 6. Flow chart for Processing CoBase queries.

profile, query context, and relaxation control operators as defined in Section 4.1. The Relaxation Manager combines those factors via certain policies (minimizing search time or nearness, for example) to restrict the search for approximate answers. We allow the input query to be annotated with control operators to help guide the mediator in query relaxation operations.

If control operators are used, the Relaxation Manager selects the condition to relax in accordance with the requirements specified by the operators. For example, a "relaxation-order" operator will dictate "relax location first, then runway length." Without such user-specified requirements, the Relaxation Manager uses a default relaxation strategy by selecting the relaxation order based on the *minimum coverage rule*. Coverage is defined as the ratio of the cardinality of the set of instances covered by the entire TAH. Thus, *coverage* of a TAH node is the percentage of all tuples in the TAH which are covered by the current TAH node. The minimum coverage rule always relaxes the condition which causes the minimum increase in the scope of the query, which is measured by the coverage of its TAH node. This default relaxation strategy attempts to add the smallest number of tuples possible at each step, based on the rationale that the smallest increase in scope is likely to generate the close approximate answers. The strategy for choosing which condition to be relaxed first is only one of many possible relaxation strategies; the Relaxation Manager can support other different relaxation strategies as well.

Let us consider the following example of using control operators to improve the relaxation process. Suppose a pilot is searching for an airport with an 8,000 feet runway in Bizerte but there is no airport in Bizerte that meets the specifications. There are many ways to relax the query in terms of location and runway length. If the pilot specifies the relaxation-order to relax the location attribute first, then the query modification generalizes the location 'Bizerte' to 'NW_Tunisia' (as shown in Figure 1) and specializes the locations 'Bizerte', 'Djedeida', 'Tunis', and 'Saminjah', thus broadening the search scope of the original query. If, in addition, we know that the user is interested only in the airports in NW and SW Tunisia and does not wish to shorten the required runway length, the system can eliminate the search in East Tunisia and also avoid airports with short and medium runways, as shown in Figure 7. As a result, we can limit the query relaxation to a narrower

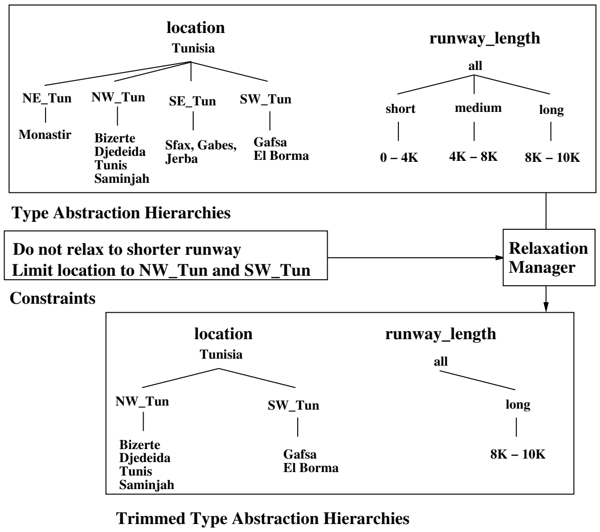

Figure 7. TAH trimming based on relaxation control operator.

scope by trimming the type abstraction hierarchies, thus improving both the system performance and the answer relevance.

5.2.2. Spatial Relaxation and Approximation

In geographical queries, spatial operators such as 'located,' 'within,' 'contain,' 'intersect,' 'union,' and 'difference' are used. When there are no exact answers for a geographical query, both its spatial and non-spatial conditions can be relaxed to obtain the approximate answers. CoBase operators also can be used for describing approximate spatial relationships. For instance, "an aircraft-carrier is *near* seaport Sfax." *Approximate spatial operators*, such as 'near-to' and 'between' are developed for the approximate spatial relationships. Spatial approximation depends on *contexts* and *domains* [21], [24]. For example, a **hospital** *near to* LAX is different from an **airport** *near to* LAX. Likewise, the *nearness* of a hospital in a **metropolitan** area is different from the one in a **rural** area. Thus spatial conditions should be relaxed differently in different circumstances. A common approach to this problem is the use of pre-specified ranges. This approach requires experts to provide such information for all possible situations, which is difficult to scale up to larger applications or to extend to different domains. Since TAHs are user and context sensitive, they can be used to provide context-sensitive approximation. More specifically, we can generate TAHs based multi-dimensional spatial attributes (MTAHs).

Further, MTAH (based on latitude and longitude) is generated based on the distribution of the object locations. The distance between nearby objects is context-

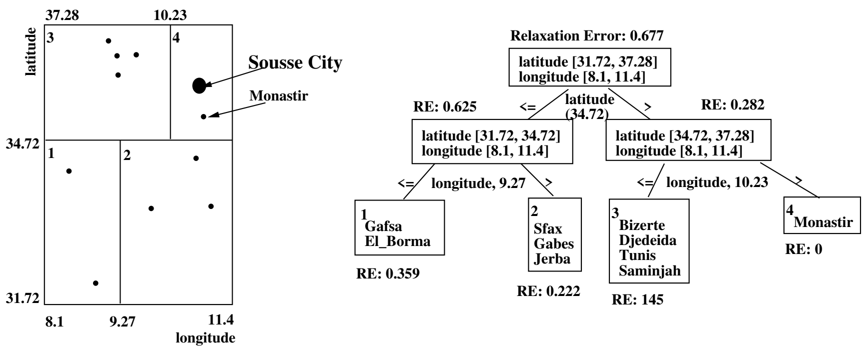

Figure 8. An MTAH for the airports in Tunisia and its corresponding two-dimensional space.

sensitive: the denser the location distribution, the smaller the distance among the objects. In Figure 8, for example, the default neighborhood distance in Area 3 is smaller than the one in Area 1. Thus, when a set of airports are clustered based on their locations, the ones in the same cluster of the MTAH are much closer to each other than to those outside the cluster. Thus, they can be considered 'near-to' each other. We can apply the same approach to other approximate spatial operators, such as 'between' (*i.e.*, a cluster 'near-to' the center of two objects). MTAHs also can be used to provide context-sensitive query relaxation. For example, consider the query: "Find an airfield at the city Sousse." Since there is no airfield located exactly at Sousse, this query therefore can be relaxed to obtain approximate answers. First, we locate the city Sousse with latitude 35.83 and longitude 10.63. Using the MTAH in Figure 8, we find Sousse is covered by Area 4. Thus, the airport Monastir is returned. Unfortunately, it is not an airfield. So the query is further relaxed to the neighboring cluster–the four airports in Area 3 are returned: Bizerte, Djedeida, Tunis, and Saminjah. Since only Djedeida and Saminjah are airfields, these two will be returned as the approximate answers.

MTAHs are automatically generated from databases by using our clustering method that minimizes relaxation error [3]. They can be constructed for different contexts and user type. For example, it is critical to distinguish a friendly airport from an enemy airport. The use of a MTAH for friendly airports restricts the relaxation only within the set of friendly airports, even though some enemy airports are geographically nearby. This restriction significantly improves the accuracy and flexibility of spatial query answering. The integration of spatial and cooperative operators provides more expressiveness and context-sensitive answers. For example, the user is able to pose such queries as, "find the airports similar-to LAX and near-to City X." When there are no answers available, both 'near-to' and 'similar-to' can be relaxed based on the user's preference (*i.e.*, a set of attributes). To relax 'near-to', airports from neighboring clusters in the MTAH are returned. To relax 'similar-to' the multiple-attribute criteria are relaxed by their respective TAHs.

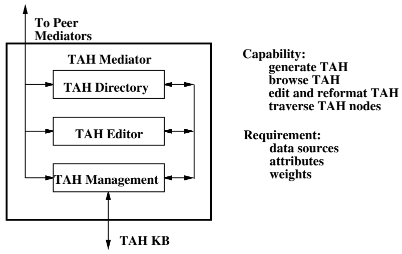

Figure 9. The TAH Mediator

Cooperativeness in geographic databases was studied in [17]. A rule-based approach is used in their system for approximate spatial operators as well as query relaxation. For example, they define that: "P is *near* to Q iff the distance from P to Q is less than *n*length_unit*, where *length_unit* is a context dependent scalar parameter, and *n* is a scalar parameter that can be either unique for the application and thus defined in domain model, or specific for each class of users and therefore defined in the user models." This approach requires *n* and *length_unit*) be set by domain experts. Thus, it is difficult to scale up. Our system uses MTAHs as a representation of the domain knowledge. The MTAHs can be generated automatically from databases based on contexts and provide a structured and context-sensitive way to relax queries. As a result, it is scalable to large applications. Further, the relaxation error at each node is computed during the construction of TAHs and MTAHs. It can be used to evaluate the quality of relaxations and to rank the nearness of the approximate answers to the exact answer.

5.3. TAH Mediator

The CoBase TAH Mediator (Figure 9) provides three conceptually separate, yet interlinked, functions to peer mediators. These are the TAH Directory, the TAH Management, and the TAH Editing facilities.

Usually, a system contains a large number of TAHs. In order to allow other mediators to determine which TAHs exist within the system and the characteristics of those TAHs, the TAH Mediator contains an intelligent directory. This directory serves to link the characteristics, context, and user type with the TAHs themselves. This directory is searchable by characteristics, user type, context, name, or any combination thereof, enabling peer mediators to locate TAHs even if they have only a partial picture of the TAH they desire. Further, the TAH directory is capable of approximate matching, so if the exact TAH a client desires is unavailable in the system, the TAH Directory lists those TAHs which may be useful.

The TAH Management facility provides client mediators with TAH traversal functions (e.g. specialization and generalization), data extraction functions (for reading the information out of TAH nodes), and formatting functions (to eliminate the need for peer mediators to understand the varied internal formats of different types of TAHs). These capabilities present a common interface, so that peer mediators can traverse and extract data from a TAH without knowing about the particular type or structure of the TAHs they are using.

The TAH Mediator supports a TAH editor which allows users to edit TAHs to suit their specific needs. The editor provides the following editing operations.

- **Delete:** to improve search efficiency, irrelevant parts of a hierarchy for a given application should be deleted.
- **Move:** if a TAH is improperly formed for its application, then a domain expert can adjust it by moving sub-hierarchies and attaching them to the appropriate places.
- **Add:** if a TAH does not provide enough details for the particular application, then additional information can be added to the hierarchy.

The TAH editor handles recalculation of all information contained within TAH nodes (e.g. the coverage and relaxation error) during the editing process and supports exportation and importation of entire TAHs if a peer mediator wishes to modify a TAH itself.

5.4. Association Mediator

Often it is desirable to provide additional information relevant to, though not explicitly stated in, a user's query. For example, in finding the location of an airport satisfying the runway length and width specifications, an association mediator (Figure 10) can provide additional information about the runway quality and weather condition so that this additional information may help the pilot select a suitable airport to land his aircraft. Thus, association needs to identify the dependencies and relationships between data distributed in multiple classes or even multiple databases. Since objects have different relationships in different problem domains, it is necessary to focus on the localized contexts in which the objects participate. Domain knowledge is then used to interpret the relationships between these objects. As an object can have multiple super-types, there exist different views of type abstraction for different problem contexts. Therefore, TAHs express relationships at the context level.

Association in CoBase is executed as a multi-step post-process. After the query is executed, the answer set is gathered with the query conditions, user profile, and application constraints. This combined information is matched against query cases from the case base to identify relevant associative information [9]. The query cases can take the form of a CoBase query which can include any CoBase construct, such as conceptual conditions (*e.g.*, runway_length_ft = short) or explicitly cooperative operations (city *near-to* 'Bizerte').

For example, consider the query

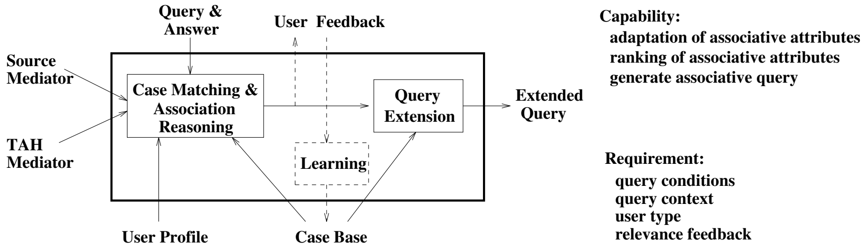

Figure 10. Association Mediator

Query Answer		Associative Information	
name	runway_length	runway_condition	weather
Jerba	9500	Damaged	Sunny
Monastir	6500	Good	Foggy
Tunis	8500	Good	Good

Figure 11. Query Answer and Associative Information for the Selected Airports.

```
SELECT name, runway_length_ft
FROM airports
WHERE runway_length_ft > 6000
```

Based on the combined information, associative attributes such as 'runway conditions' and 'weather' are derived. The associated information for the corresponding airports is retrieved from the database and then appended to the query answer, as shown in Figure 11. Since association can be transitive, we use Case Based Reasoning that based on past queries, user profile, and query context to terminate the association.

A Case Memory (see Figure 12) consists of *cases* and *association links*. Cases are past user queries (*e.g.*, Q_1, Q_2, Q_3, or Q_4). An association link (*e.g.*, l_1) is established by the attributes shared by the two cases (*e.g.*, Q_1 and Q_2), and the corresponding weight (*e.g.*, w_1) represents the usefulness of the association between the two cases. When a user query, Q_{user}, is executed, its conditions, user type, and query context are *compared* against the Case Memory for similar cases (*e.g.*, Q_1 and Q_2 are similar to Q_{user}). Based on the set of similar cases, a set of association subjects (Q_4 and Q_3) can be *selected* through the traversal of association links (l_2, l_3, l_4). The usefulness of an association is *computed* from the similarity measure of the corresponding cases and the weights of the traversed association links. The cases with the high usefulness values (*e.g.*, Q_3 and Q_4) are *adapted* into the user query (Q'_3 and Q'_4) as associations to the user. Initially, the Case Memory has not acquired any experience. *User feedback* on the usefulness of the associations is incrementally integrated into the Case Memory by adjusting

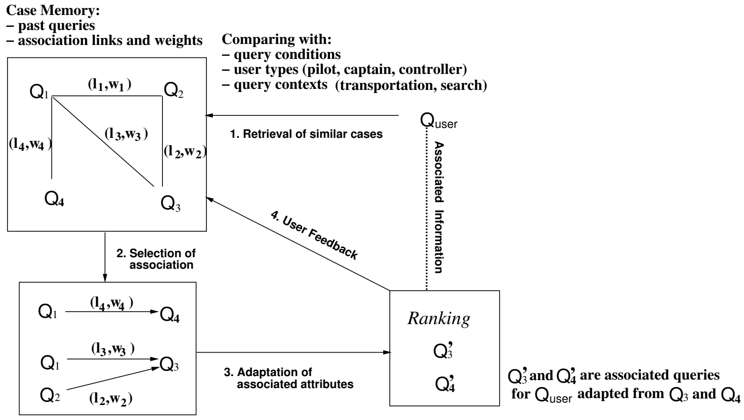

Figure 12. Case Memory and Associative Links.

the weights of the traversed association links. In this way, the Case Memory can accumulate experience from the user feedback.

Our current Case Base, consisting of about 1500 past queries, serves as the knowledge server for the association module. The size of the Case Base is around 2MB. For association purposes, we use the 300-case set which is composed of past queries used in the transportation domain. For testing performance and scalability of the system, we use a 1500-case set which consists of randomly generated queries based on user profile and query template over the transportation domain. Users can also browse and edit association control parameters such as the number of association subjects, associated links and weights of a given case, the threshold for association relevance, *etc.*

5.5. Explanation Mediator

Query relaxation and association involve inference over vast data and knowledge sources. If users are to trust derived answers, it is vital to provide explanations of these processes. To not overwhelm the user, explanations should initially be summary. If required, users should be able to interactively obtain further descriptions, definitions, or justifications. Based on the user and context, explanations should occur at appropriate times during processing. One extreme is the system running automatically, only summarizing its work once it has completed. Another extreme is the system running in detail, explaining each action, giving the user the ability to monitor progress. Explanations should also be tailored to a user's understanding of the system. We have developed an explanation mediator [20] to serve these purposes.

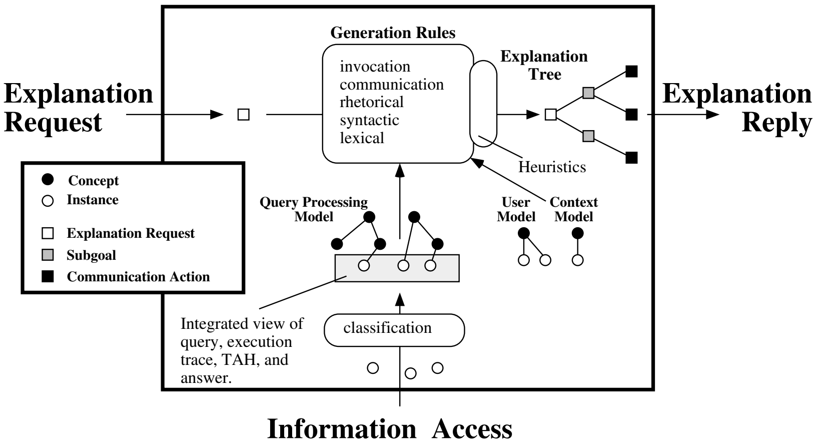

Figure 13. Explanation Mediator

The explanation mediator is *capable* of providing **removed multi-media** interactive, user-sensitive, and context-dependent explanations of CoBase cooperative operations. A client sends the explanation mediator a simple explanation request and receives back an explanation consisting of natural language and recommended visualizations. The client presents the explanation on their GUI and a simple protocol enables the user to interact with this initial explanation, causing the explanation mediator to generate further explanations. The explanation mediator *requires* access to queries, TAHs, execution traces, and answers (see Figure 13). CoBase mediators make this information available to the explanation mediator and the explanation mediator maintains a model of query processing, consisting of concepts and classification rules, by which it interprets such information. This approach is similar to the Explanation Explainable Expert System (EES) [25] framework, though its model of query processing is built specifically to interpret CoBase operations, and work is focused on providing explanations any time during system execution. **changed In addition, the representation, classification and generation formalism covers the complete explanation generation task, thus easing integration, maintenance and extension.**

The explanation mediator represents information such as queries, execution traces, TAHs, and answers in a representation similar to a semantic network. The particular representation, a directed hyper-graph, offers several advantages over standard semantic networks for explanation. Most importantly our hyper-graph allows for abstraction on the label linking nodes (instances, values, or concepts), enabling specific to generic access of the graph. The hyper-graph also enables the direct expression of one-to-many and many-to-many type relationships. Concepts in this graph (e.g. Query, Relax-Query-Action, TAH-node, etc.) include those of the query processing model. As the relaxation and association mediators execute, they send a stream of information that is instantiated in this graph. Classification rules in the query processing model interpret and augment this information as it added to the graph. In addition, the graph contains a user and context-model, enabling the generation of user-sensitive, context-dependent explanations.

The explanation mediator accepts an explanation request on a node (or set of nodes) in the graph and produces an explanation tree. This tree is created via the depth-first application of generation rules toward solving the initial explanation request. The explanation request is at the root of the explanation tree, while the text and recommended visualizations of the explanation are leaves in the tree. Intermediate nodes are the subgoals carried out to achieve the root explanation request. Generation rules consist of a goal, a set of constraints, and a set of actions. If the explanation request (or an intermediate subgoal) is matched by a generation rule goal, and if all of the rule's constraints are met, then the rule's actions are performed. These actions may either be subgoals that are in turn solved by application of generation rules or may be primitive communication actions (e.g. text or a visualization call).

A single inference mechanism applies classification and all levels of generation rules. Classification rules interpret CoBase information as instances of query processing concepts. Generation rules expand explanation requests and intermediate nodes during the generation of an explanation tree. There are five types of generation rules: invocation, communication, rhetorical, syntactic and lexical. Invocation rules determine what should be explained under various user models and contexts. Communication rules determine what information will be expressed. Rhetorical rules determine the manners in which information is expressed. Syntactic rules constrain text to valid English. Lexical rules determine the names that will be given to CoBase actions and objects. Invocation and communication rules are typically the rules that match explanation requests while rhetorical, syntactic, and lexical rules typically expand intermediate nodes and produce primitive communication actions.

Different users require different types of explanation under different contexts. This is accounted for by the fact that more than one generation rule may match to expand a node in the explanation tree. Alternate generation rules vary in which aspects of CoBase's process or results should be expressed and to what depth. Heuristics control how an explanation is generated by controlling which rule among the matching rules will fire to expand a goal. Note that these heuristics apply only to generation rules which determine when and how to express information . The interpretation of the relaxation and association processes is non-heuristic.

The quality of explanations (with respect to completeness, correctness, and precision) depends on access to CoBase queries, execution traces, TAHs, and answers, and also on the effort expended in populating and refining the explanation rule-base. Elapsed times depend on access cost to such information and the computational complexity of synthesizing the explanation tree. The dominant cost is the synthesis of explanation trees, particularly the cost of matching generation rules to expand nodes. A RETE pattern-matcher [11] performs these matches. Performance scales as the logarithm of the number of generation rules. The explanation mediator is implemented in C++ and CLIPS. There are approximately **changed 60 concepts** and 20 classification rules in the query processing model. There are approximately 80 general generation rules for explanation of CoBase cooperative operations. In two CoBase application domains, electronic warfare and transportation planning, we have extended these general generation rules to create 40 specific generation rules for the electronic warfare domain and 30 rules for the transportation planning domain. These domain specific generation rules were based on simple revisions of the general generation rules. The explanation mediator can usually produce explanations in under a second on a Sun SPARC 10 workstation.

Current work is focused on providing a method by which system administrators and users may interactively **refine and extend** explanations. Currently, programmers must write new generation rules. Instead, administrators, through a sequence of editing operations, should be capable of **changed extending** initially general CoBase explanations into precise, domain specific forms. By doing so, the **changed generation** rule-base is populated with domain specific rules. Through feedback and interaction, users should be able to refine explanations by altering the heuristic knowledge that determines which rules are selected.

5.6. Data Source Mediator

The Data Source Mediator (DSM) provides two services (Figure 14):

1. *Virtual Database Interface.* Different data sources usually have different schema. To extend CoBase to different data sources, a virtual database interface is developed based on a uniform data source representation. The virtual database interface represents a data source as a list of tables (each table consists of a list of attributes). DSM extracts schema information from a specific data source and builds the virtual database interface. As a result, data schema and data can be accessed from data sources without the need of knowing the differences of the underlying data sources. The *Query and Schema Translation* module converts a virtual database information request into a specific request supported by one of its underlying data sources.

2. *Query Conversion.* Different data sources may also use different query languages. When a query is processed in CoBase, it uses an internal representation called *query object.* To pose a processed query to a specific data source, DSM must convert query objects into the specific query language for the target data source (*e.g.*, SQL).

In DSM, each data source has a database interface to handle the specific requirements. Currently, we have database interfaces for Oracle (a relational database), RDB (a UNIX-based text database), and Objectivity (an object-oriented database). New data sources can be incorporated by adding a new interface module to

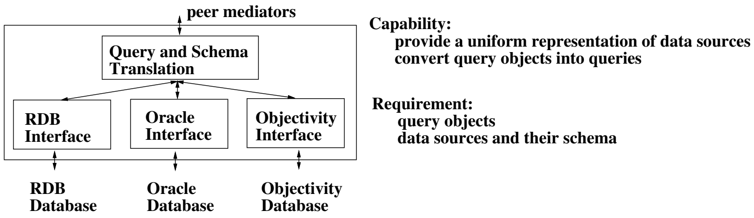

Figure 14. The Data Source Mediator

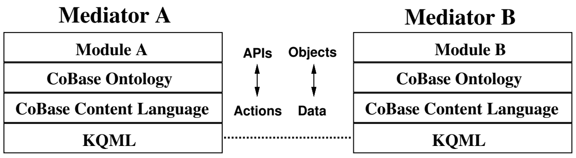

Figure 15. Flow of mediator communications.

the DSM, which does not affect the rest of the system. Thus, DSM makes CoBase scalable and extensible to different types of data sources.

5.7. Inter-Mediator Communications

Since CoBase mediators are independent processes possibly running on different computers, a communication language is needed to facilitate information exchange and service request among these mediators. Each CoBase mediator (Figure 15) consists of a *CoBase module* (*e.g.*, A or B), the *CoBase Ontology Layer* (for message interpretation), the *CoBase Content Language Layer* (for message representation), and *the KQML Layer* (for message transportation):

1. A CoBase module performs a specific cooperative service, such as relaxation, association, explanation, or TAH management. Each module has a set of *APIs* and represents its information in C++ *objects*. The services of a module are invoked through its APIs. Since CoBase mediators are running on separated process spaces, an API call must be transported to the module supporting it. When APIs or objects are represented in a transportable format, they are called *CoBase messages*.

2. A CoBase Ontology (CO) provides basic vocabulary to compose CoBase messages for mediator communications. In a CoBase message, an API call is represented as an *action* and an object is represented as a *data*. For example, an API call for relaxing a given condition:

   ```
   (generalize_condition TAH_NODE *tah_node)
   ```

is represented in CCL as

```
(generalize_condition "TAH_NODE_REFERENCE")
```

CO functions as a dictionary for the translation between APIs/objects and actions/data.

3. A CoBase Content Language (CCL) is used to represent CoBase messages in a structured form so that they can be interpreted and understood among different mediators. CoBase mediators require interactive communication capabilities. For example, an explanation returned by the explanation mediator contains some actions for further interaction. A user can select on certain phrases of the explanation to require the explanation mediator for further clarification, elaboration, or justification. Thus, data exchanged among CoBase mediators not only contain necessary information for the completion of an action, but also have *embedded* actions for further interaction. CCL is designed to support such capability.

4. The Knowledge and Query Manipulation Language (KQML) [12] serves as envelopes to pass CoBase messages among the CoBase mediators across the network.

Among the three components of the Communication Language, the Ontology (*i.e.*, CO) is developed based on CoBase modules and their APIs, while the content language (*i.e.*, CCL) and the transport language (*i.e.*, KQML) are CoBase-independent.

The flow of CoBase mediator communication is illustrated in Figure 15. The communication starts with a request for action from a mediator (*e.g.*, A). The request (*i.e.*, an API call) and its parameters (*i.e.*, C++ objects) are translated into action and data in CCL by CO, wrapped inside a KQML message, and passed to a designated mediator (*e.g.*, B). After the message is received, it is unwrapped at the KQML Layer and translated back to an API call by CO. The call is performed by B and an answer is returned. The answer (in C++ object format) is translated into CCL and sent back to A, in the same manner as as it does from A to B.

The following example describes the use of CoBase messages for communication between the Relaxation Mediator (RM) and the TAH Manager (TM). TM provides TAH traversing operations such as generalization and specification. When RM needs to relax a condition via a TAH node with identifier "`TAH_NODE_ID_113`", it sends a request (*i.e.*, an API call) to TM:

```
(generalize_condition TAH_NODE_ID_113)
```

This call is translated into a CCL action:

```
(generalize_condition "TAH_NODE_ID_113")
```

This message is wrapped inside a KQML performative and passed to TM:

```
(achieve :language CoBaseContentLanguage
         :ontology CoBaseOntology
         :reply-with m_17
         :sender RelaxationMediator
         :receiver TahManager
         :content (generalize_condition "TAH_NODE_ID_113"))
```

TM returns relaxed conditions to RM:

```
"'BIZERTE' 'TUNIS' 'DJEDEIDA'"
```

Another example to illustrate the communication between RM and the Explanation Mediator (EM): a user requires the description of a relaxation process identified by `"relaxQueryAction-01"`.

```
(explain "relaxQueryAction-01" "describe")
```

EM returns an explanation to RM: "**The query** retrieving **airport name** where **runway length** is greater than 2000 feet, ... after relaxation **location name Bizerte** relaxed to **location names Tunis, Djedeida, ...**" (phrases in **bold** can be selected on for further explanation). The corresponding CCL is:

```
(DATA Explanation
      ("The query" :action (explain "node-ref-1")
       "retrieving"
       "airport name" :action (explain "node-ref-2")
       "where"
       "runway length" :action (explain "node-ref-3")
       "is greater than 2000 feet"
       ...)
```

"`node-ref-1`," "`node-ref-2`," and "`node-ref-3`" are node identifiers in the explanation graph generated by EM for "`relaxQueryAction-01`." When a user selects a phrase which has a predefined action (*e.g.*, **The query**, **airport name**), the selected action is sent back to EM to perform the action and returns more explanation on the phrase (*i.e.*, **The query**, or **airport name**).

6. Implementation

6.1. Object-Oriented Implementation

We employ the object-oriented paradigm to implement CoBase. In this implementation, queries, database schema/interface, cooperative operators, and TAHs are represented as objects. Because of *polymorphism* and *encapsulation*, differences among various specific query languages, databases, cooperative functions, or types of TAHs are hidden behind abstract objects. Thus, query relaxation and other cooperative modules can be built upon these abstract objects with little knowledge about the implementation of the specific objects. For example, we have different objects for different databases including relational databases (*e.g.*, Oracle or SyBase object) and object-oriented databases (*e.g.*, Objectivity object). Each provides its own interface for database access. But CoBase modules only see a *database* object which has an abstract interface to access any database. This implementation enables these modules to work easily with different databases. Moreover, the object-oriented approach provides easy enhancements to CoBase. Adding a new cooperative function, for instance, simply entails defining a new cooperative operator object. The modification effects are localized and minimized through abstraction. Further, *inheritance* allows code sharing and reusing, thus reducing the development and maintenance effort. For example, functions common to all databases can be implemented in the abstract database object and inherited by all specific database objects. In this object-oriented paradigm, CoBase consists of multiple reusable modules. These modules are built on top of abstract objects. Thus, they can be easily ported to other systems by building *host-compliant* objects.

6.2. Graphical User Interfaces

CoBase's control mechanisms include graphical user interfaces for the Relaxation Mediator, Associative Mediator, and the Explanation Mediator. The GUI for the relaxation was implemented in C++ and X/MOTIF. It enables users to browse the database, visualize the query relaxation process, and view the answers. Further, a map server was integrated into CoBase. It allows a user to pose queries based on geographic objects such as countries, states, cities, airports, rivers, *etc.* An important feature of these objects is that they can be *spatially* related via 'located,' 'within,' 'contain,' 'intersect,' 'union,' and 'difference.' These spatial operators can be used in our system to specify spatial conditions in spatial queries. For example, a query such as, "find all the airports in the region specified on the map" retrieves the airports in the user-specified region and displays the airport objects on the map, as shown in Figure 16. More importantly, it allows a user to ask complex queries by the simple 'point-and-click' method. The GUI for the explanation supports a hypertext-like browser for explanation. It allows users to interactively ask for more definition, elaboration, justification, or simply summarization.

7. Performance Evaluation

In this section, we present the CoBase performance based on measuring the execution of a set of queries on the CoBase testbed developed at UCLA for the ARPI transportation domain. The performance measure includes response time for query relaxation, association, and explanation, and the quality of answers. The response time depends on the type of queries (*e.g.*, size of joins, number of joins) as well as the amount of relaxation, association, and explanation required to produce an

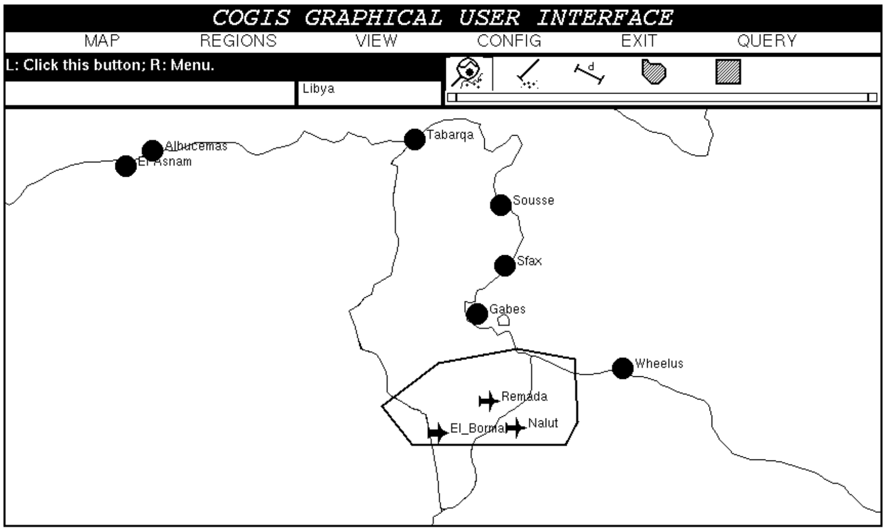

Figure 16. Find all the airports in a user-specified area.

answer. The quality of the answer depends on the amount of relaxation and association involved. The user is able to specify the relaxation and association control to reduce the response time and also to specify the requirement of answer accuracy. In the following, we shall show four example queries and their performances. The first query illustrates the relaxation cost. The second query shows the additional explanation cost, while the third query shows the additional association cost. The fourth query shows the processing cost for returned query answers as well as the quality of answers by using TAH vs MTAH for a very large database table (about 200,000 tuples).

Query 1: Find nearby airports can land C-5.

Based on the airplane location, the relaxation mediator translates 'nearby' to a pre-specified or user-specified latitude and longitude range. Based on the domain knowledge of C-5, the mediator also translates 'land' into required runway length and width for landing the aircraft. The system executes the translated query. If no airport is found, the system relaxes the distance (by a predefined amount) until an answer is returned. In the above query, an airport is found after one relaxation. Thus, two database retrievals (*i.e.*, one for the original query, one for the relaxed query) are performed. Three tables are involved: Table GEOLOC (50,000 tuples), table RUNWAYS (10 tuples), table AIRCRAFT_AIRFIELD_CHARS (29 tuples). The query answers provide airport locations and their characteristics.

```
Elapsed time:   5 seconds processing time for relaxation
                40 seconds database retrieval time
```

Query 2: Find at least 3 airports similar-to Bizerte based on runway-length and runway-width.
The relaxation mediator retrieves runway characteristics of Bizerte airport and translates the similar-to condition into the corresponding query conditions (runway length and runway width). The system executes the translated query, and relaxes the runway length and runway width according to the TAHs until at least 3 answers are returned. Noted the TAH used for this query is a Runway-TAH based on *runway-length and runway-width*, which is different from the Location-TAH based on *latitude and longitude* (shown in Figure 8). The nearness measure is calculated based on weighted mean square error. The system computes similarity measure for each answer obtained, ranks the list of answers, and presents it to the user. The system obtains five answers after two relaxations. The best three are selected and presented to the user. Two tables are involved: table GEOLOC (50000 tuples), table RUNWAYS (10 tuples).

```
Elapsed time:   2 seconds processing time for relaxation
                10 seconds database retrieval time
```

The explanation mediator generates the following description of the SQL query:

> *The query retrieves airport name, runway length, and runway width where answers are similar to the runway length and runway width of the airport name Bizerte. A minimum of three answers are required.*

```
Elapsed time:  5 seconds processing time for explanation
```

The explanation mediator generates the following description of relaxation process:

> *Query relaxed: airport name Bizerte relaxed to Bizerte and El Borma.*

```
Elapsed time:   2 seconds processing time for explanation
```

> *Query relaxed: airport name Bizerte and El Borma relaxed to airport name Bizerte, Djedeida, El Borma, Gabes and Gafsa.*

```
Elapsed time:  2 seconds processing time for explanation
```

Further, the explanation provides the following summarization:

> *airport name Bizerte relaxed to Gafsa, Gabes, El Borma, Djedeida and Bizerte yielding 5 answers. These answers are ranked by similarity to the runway length and runway width of the airport name Bizerte based on weighted mean square error measure.*

```
Elapsed time:  2 seconds processing time for explanation
```

Query 3: Find seaports in Tunisia with a refrigerated storage capacity of over 50 tons.
The relaxation mediator executes the query. The query is not relaxed, so one database retrieval is performed. Two tables are used: table SEAPORTS has 11 tuples; table GEOLOC has about 50000 tuples.

```
Elapsed time:   2 seconds processing time for relaxation
                5 seconds database retrieval time
```

The association mediator returns relevant information about the seaports. It compares the user query to previous similar cases and selects a set of attributes relevant to the query. Two top associated attributes are selected and appended to the query. CoBase executes the appended query and returns the answers to the user, together with the additional information. The two additional attributes associated are location name and availability of railroad facility near the seaports.

```
Elapsed time:  10 seconds for association computation time
```

Query 4: Find at least 100 cargos of code '3FKAK' with the given volume (length, width, height), code is non-relaxable.
The relaxation mediator executes the query, and relaxes the height, width, and length according to MTAH, until at least 100 answers are returned. The query is relaxed four times. Thus, five database retrievals are performed. Among the tables accessed, Table CARGO_DETAILS has 200,000 tuples, a very large table.

```
Elapsed time:  3 seconds processing time for relaxation using MTAH
               2 minutes database retrieval time for 5 retrievals
```

By using single TAHs (*i.e.*, single TAHs for height, width, and length respectively), the query is relaxed 12 times. Thus 13 database retrievals are performed.

```
Elapsed time:  4 seconds for relaxation by single TAHs
               5 minutes database retrieval time for 13 retrievals
```

For queries involving multiple attributes in the same relation, using an MTAH that covers multiple attributes would provide better relaxation control than using a combination of single-attribute TAHs. The MTAH compares favorably with multiple single-attribute TAHs in both quality and efficiency. We have shown that an MTAH yields a better relaxation strategy than multiple single-attribute TAHs. The primary reason is that MTAHs capture attribute-dependent relationships which cannot be captured when using multiple single-attribute TAHs.

Using MTAHs to control relaxation is more efficient than using multiple single-attribute TAHs. For the above example, relaxation using MTAHs require an average of 2.5 relaxation steps, while single-attribute TAHs require 8.4 steps. Since a database query is posed after each relaxation step, using MTAHs saves around 6 database accesses on average. Depending on the size of tables and joins involved, each database access may take from 1 second to about 30 seconds. As a result, using MTAHs to control relaxation saves a significant amount of user time.

With the aid of domain experts, the above queries can be answered by conventional databases. Such an approach takes a few minutes to a few hours. However, without the aid of the domain experts, it may take hours to days to answer these queries. CoBase incorporates domain knowledge as well as relaxation techniques. Thus, it is able to automatically search for the answer with significantly less time.

8. Technology Transfer of CoBase

CoBase stemmed from the transportation planning application for relaxing query conditions. CoBase has been linked with SIMS [1] and LIM [23] as a knowledge server for the planning system. SIMS performs query optimizations for distributed databases, and LIM provides high level language query input to the database. A Technical Integration Experiment (TIE) has been performed to demonstrate the feasibility of this integrated approach. CoBase technology is being implemented for the ARPI transportation application. In addition, CoBase has also been successfully applied to the following domains.

In electronic warfare, one of the key problems is to identify and locate the emitter for radiated electro-magnetic energy based on the operating parameters of observed signals. The signal parameters are radio frequency, pulse repetition frequency, pulse duration, scan period, etc. In a noisy environment, these parameters often cannot be matched exactly within the emitter specifications. CoBase can be used to provide approximate matching of these emitter signals. A knowledge base (TAH) can be constructed from the parameter values of previously identified signals and also from the peak (typical, unique) parameter values. The TAH provides guidance on the parameter relaxation. The matched emitters from relaxation can be ranked according to relaxation errors. Our preliminary results have shown that CoBase can significantly improve emitter identification as compared to conventional database techniques, particularly in a noisy environment. From the line of bearing of the emitter signal, CoBase can locate the platform that generates the emitter signal by using the "near-to" relaxation operator.

In medical databases that store X-rays and MR images, the images are evolution and temporal-based. Further, these images need to be retrieved by object features or contents rather than patient ID [8]. The queries asked are often conceptual and not precisely defined. We need to use knowledge about the application (*e.g.*, age class, ethnic class, disease class, bone age etc.), user profile and query context to derive such queries [5]. Further, to match the feature exactly is very difficult if not impossible. For example, if the query, "Find the treatment methods used for tumors *similar to* $X_i(location_{x_i}, size_{x_i})$ on 12-year-old Korean males," cannot be answered, then based on the TAH shown in Figure 17, we can relax "tumor X_i" to tumor "Class X," and "12-year-old Korean male" to "pre-teen Asian," which results in the following relaxed query: "Find the treatment methods used for tumor "Class X" on pre-teen Asians." Further, we can obtain such relevant information as the success rate, side effects, and cost of the treatment from the association operations. As a result, query relaxation and modification are essential

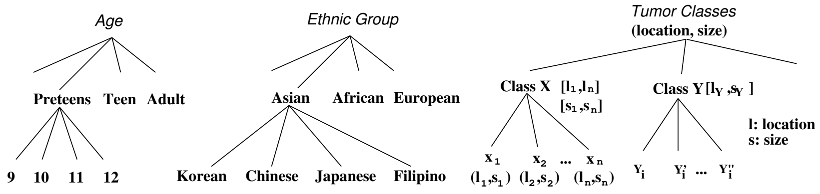

Figure 17. Type Abstraction Hierarchies for the Medical Query Example.

to process these queries. We have applied CoBase technology to medical imaging databases [18]. TAHs are generated automatically based on context-specific (*e.g.*, brain tumor) image features (*e.g.*, location, size, shape, *etc*). Once the TAHs for the medical image features have been constructed, query relaxation and modification can be carried out on the medical features.

The use of CoSQL constructs such as 'similar-to,' 'near-to,' and 'within' can be used in combination, thus greatly increasing the expressibility for relaxation. For example, we can express "`find tumors` *similar to* `the tumor x` *based-on* `(shape, size, location) and` *near to* `object O` *within* `a specific range (`*e.g.*`, angle of coverage).`" The relaxation control operators, such as matching tumor features in accordance to their importance, can be specified by *relaxation-order* `(location, size, shape)` to improve the relaxation quality.

9. Conclusions

We have presented a structured approach that uses Type Abstraction Hierarchy for providing query modification. TAHs provide multi-level knowledge representation and can be generated automatically from data sources. Further, TAHs are user and context sensitive, and easy to customize and maintain. Thus, TAHs are scalable to large systems. We have also extended the SQL to CoSQL. This extended language provides cooperative operators that allow the user to explicitly specify relaxation operations and controls. These operators are very general and can be applied to different domains. Combination of these operators in a query can greatly improve the expressive power. Based on user profile and application contexts, the relaxation manager limits the search scope and also filters out the unsuitable answers. To provide relevant information to the query answers, a pattern-based framework is used for deriving associative (relevant) information from past cases. An explanation facility is included which summarizes the query modifications and relaxation process, and also provides the nearness of the approximate answer to the exact answer. Further, a cooperative geographic information system has been developed to provide information retrieval and analysis based on geographic objects (*e.g.*, countries, cities, airports) and spatial relations (*e.g.*, 'located,' 'within,' 'between,' 'near-to,' 'closest'), and to perform context-sensitive relaxation of geographic relations. CoBase is implemented in a mediator-based architecture which is scalable and extensible.

Performance measurements on our testbed reveal that the cost for relaxation, explanation, and association is fairly small. The major cost is due to database retrieval which depends on the number of relaxations required before obtaining a satisfactory answer. For queries with multiple dependent attributes, the use of MTAH can reduce the number of relaxations, thus decreasing the overall cost.

CoBase stemmed from the ARPA/Rome Lab Planning Initiative for providing approximate query answers from a large and heterogeneous database. It has also been applied in a medical imaging database (X-ray, MRI) for approximate matching of image features and contents, and in electronic warfare for approximate matching of emitter signals (based on a set of parameter values) and also for locating the platforms that generate the signals via spatial relaxation.

10. Acknowledgment

The research and development of CoBase has been a team effort. We would like to acknowledge the past and present CoBase members, Guogen Zhang, Ladislav Berkovich, Jason Robins, Henrick Yau, Gilles Fouques, Matthew Merzbacher, Frank Meng, and Brad Perry for their contributions in design and implementation efforts.

Notes

1. The definition of abstraction allows successive single-value abstractions, which does not provide real abstraction, therefore in implementation, they are collapsed into a single abstraction.
2. The goodness is defined by the intra-class similarity and inter-class dissimilarity.
3. A cut c is a value that separates a cluster of numbers $\{x|a \leq x \leq b\}$ into two sub-clusters $\{x|a \leq x \leq c\}$ and $\{x|c < x \leq b\}$.
4. Dependency here means that all the attributes as a whole define a coherent concept. For example, the length and width of a rectangle are said to be "semantically" dependent. This kind of dependency should be distinguished from the *functional dependency* in database theory.

References

1. Y. Arens and C. Knoblock. Planning and reformulating queries for semantically–modelled multidatabase systems. In *Proceedings First International Conference on Information and Knowledge Management (CIKM)*, pages 92–101, Baltimore, Maryland, 1992.
2. Wesley. W. Chu and Q. Chen. A structured approach for cooperative query answering. *IEEE Transactions on Knowledge and Data Engineering*, 6(5):738–749, 1994.
3. Wesley W. Chu and Kuorong Chiang. Abstraction of high level concepts from numerical values in databases. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, July 1994.
4. Wesley W. Chu, Qiming Chen, and Rei Chi Lee. Cooperative query answering via type abstraction hierarchy. In S.M. Deen, editor, *Cooperating Knowledge Based Systems*, pages 271–292. Springer-Verlag Inc., October 1991.

5. Wesley W. Chu, A.F. Cardenas, and R.K. Taira. KMeD: A knowledge-based multimedia medical distributed database system. *Information Systems*, 20(2):75–96, 1995.
6. F. Cuppens and R. Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. In *Proceedings of the 2th International Conference on Expert Database Systems*, Virginia, USA, 1988.
7. F. Cuppens and R. Demolombe. How to recognize interesting topics to provide cooperative answering. *Information Systems*, 14(2):163–173, 1989.
8. W.W. Chu, I.T. Ieong, and R.K. Taira. A semantic modeling approach for image retrieval by content. *Journal of Very Large Database Systems*, 3:445–477, October 1994.
9. G. Fouque, W.W. Chu, and H. Yau. A case-based reasoning approach for associative query answering. In *Proceedings of the Eighth International Symposium on Methodologies for Intelligent Systems*, Charlotte, NC, October 1994.
10. D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.
11. C.L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem, 1982.
12. T.W. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, J. Pelavin, S. Shapiro, and C. Bech. Specification of the KQML agent-communication language, plus example agent policies and architectures (draft). prepared by the ARPA Knowledge Sharing Initiative External Interfaces Working Group, June 1993.
13. M. A. Gluck and J. E. Corter. Information, uncertainty, and the unity of categories. In *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, pages 283–287, Irvine, CA, 1985.
14. T. Gaasterland, P. Godfrey, and J. Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1:123–157, 1992.
15. T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform of cooperative answering. *Journal of Intelligent Information Systems*, 1:293–321, 1992.
16. A.S. Hemerly, M.A. Casanova, and A.L. Furtado. Exploiting user models to avoid misconstruals. In R. Demolombe and T. Imielinski, editors, *Nonstandard Queries and Nonstandard Answers*, pages 73–98. Oxford Science Publications, 1994.
17. A.S. Hemerly, A.L. Furtado, and M.A. Casanova. Towards cooperativeness in geographic databases. In *Proc of the 4th International Conference on Database and Expert Systems Applications*, Prague, Czech Republic, 1993.
18. H.K. Huang and R.K. Taira. Infrastructure design of a picture archiving and communication system. *American Journal of Roentgenology*, 158, 1992.
19. Matthew Merzbacher and Wesley W. Chu. Pattern-based clustering for database attribute values. In *Proceedings of AAAI Workshop on Knowledge Discovery*, Washington, DC, 1993.
20. M. Minock and W.W. Chu. Explanation for cooperative information systems. In *Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems*, Baltimore, MD, 1996.
21. D.M. Mark and A.U. Frank. Concepts of space and spatial language. In *Proceedings of the 9th International Symposium on Computer-Assisted Cartography*, pages 538–556, Baltimore, MD, April 1989.
22. A. Motro. VAGUE: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, 1988.
23. D. P. McKay, J. Pastor, and T. W. Finin. View-Concepts: Knowledge-based access to databases. In *Proceedings First International Conference on Information and Knowledge Management (CIKM)*, pages 84–91, Baltimore, Maryland, 1992.
24. R. Subramanian and N.R. Adam. Ill-defined spatial operators in geographic databases: their nature and query processing strategies. In *Proceedings of ACM Workshop on Advances in Geographical Information System*, pages 88–93, Washington, DC, October 1993.

25. W. Swartout, C. Paris, and J. Moore. Design for explainable expert systems. *IEEE Expert*, 6(3):58–64, June 1991.
26. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.