Computer Science Department Technical Report

University of California

Los Angeles, CA 90024-1596

COOPERATIVE QUERY ANSWERING VIA TYPE ABSTRACTION HIERARCHY

Wesley W. Chu
Qiming Chen
Rei-Chi Lee

October 1990
CSD-900032

Cooperative Query Answering via Type Abstraction Hierarchy

Wesley W. Chu, Qiming Chen and Rei-Chi Lee

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

ABSTRACT

Cooperative query answering consists of analyzing the intent of the query and providing generalized, neighborhood or associated information relevant to the query. The key issues to accomplish cooperative query answering consist of supporting different knowledge representations at different abstract levels and providing inference between these levels. In this paper the *Type Abstraction Hierarchy* is proposed which is characterized by dealing with subtyping from subsumption, composition, and also abstraction views. Based on the type abstraction feature provided by this model, an inference technique for cooperative query answering is developed. Such an inference is performed by *abstracting* and *refining* the goal to generalize and specialize the query scope and to derive relevant answers with different generality, coverage and approximation, or to link related subjects at certain levels, by using different representations of knowledge given at different levels. A prototype system has been implemented at UCLA that demonstrates the use of this approach for certain decision making and problem solving applications such as conceptual query processing, neighborhood inference, and subject association.

This research is supported by DARPA contract F29601-87-C-0072 and ONR contract N00014-88-K0434.

1. Introduction

Traditional query processing provides exact answers to queries. It usually requires that users fully understand the database structure and content to issue a query. Due to the complexity of the database applications, incorrect queries are frequently issued and the users often receive no answers. **Cooperative query answering** is a process of analyzing the intent of the query and providing generalized, approximate, or associated information that is relevant to the query. In a cooperative query answering environment, queries can be imprecisely specified and accessing data is not limited to a single answer but can obtain relevant information of a wider scope, or even approximate information when the exact answer is not available. Thus, a cooperative query answering system may respond to a query by providing

(a) neighborhood information,
(b) general information,
(c) associative information.

In order to facilitate cooperative query answering, it is necessary to associate one concept to another by taking into account the related factors. In fact, **association** is one of the essential intelligent mechanisms used to make decisions or solve problems. Indeed this notion has been studied in various semantic data modeling approaches [SMI77] [HAM78] [BRO81] [SU83] [AL90]. The semantic links provided by those data models can considerably relax users from making their own programs to handle the relationships between database objects.

Equally important to association is the **knowledge level** where the association is represented. Since many complex data intensive problems do not have clean representations at lower knowledge levels, we often need to describe semantic links at certain higher knowledge levels in terms of more abstract object representations.

The use of maps is an evident example. A geographic object, such as "Los Angeles" city, has different representations on differently scaled maps. On a small scaled map it is shown just as a circle symbol, but on a larger scaled map it is represented as a graph. Different scaled maps are representations of geographic knowledge at different abstract levels. To solve a problem it is usually necessary to move the attention between these levels. For example, if we plan to go to a place in San Francisco from a place in Los Angeles, we shall first move to a higher topological level. We would find the Freeway 101 as the connection between Los Angeles and San Francisco, since the knowledge "Freeway 101 connects Los Angeles and San Francisco" is represented at such a level. On the contrary, it is not reasonable to replace this knowledge by a large number of lower level rules listing all the particular places in Los Angeles and San Francisco that can be connected by the Freeway 101.

The above example shows the fact that *associations between related concepts are easily represented and traced at certain higher knowledge levels.* We can easily find more examples indicating this fact. For instance, "system problem" can be viewed as the abstract representation of a number of problems such as "disk writing problem", "printing problem" and so on, it is efficient to give the rule at such an abstract level for relating "system problem" with "system manager", such as

"If there is a *system problem* then call the *system manager*"

rather than stating a number of specialized rules as

"If there is a *disk writing problem*" then call the "system manager".
"If there is a *printing problem*" then call the "system manager".

:

As described thus far, there are two key issues in providing cooperative query answering : the *semantic associations* and the *multi-level knowledge representation.* The first issue has been addressed in semantic data modeling, type hierarchy, object-orientation, ..., etc. However, the second issue was rarely addressed previously. Although the notion of subsumption-based type hierarchy allows objects of a class to be treated as the members of its super class at a higher knowledge level, besides changing the type name it does not facilitate different instance values to accommodate the knowledge representations at different knowledge levels. In fact, this issue was generally omitted by previous semantic data modeling approaches and object-oriented database approaches. This motivates us to develop the type abstraction hierarchy to furnish multi-level knowledge representation.

In this paper we shall first develop the notion of **Type Abstraction**, and propose the **Type Abstraction Hierarchy** to integrate the subsumption-based, composition-based, and abstraction-based type hierarchy notion for supporting the multi-level knowledge representations. Next, using the type abstraction hierarchy, we shall present an inference technique that is based on *query rewrite* to allow reasoning between different knowledge levels thus enabling us to provide cooperative query answering in a systematic way. Finally we shall give examples to demonstrate the proposed methodology for cooperative query answering.

2. The Notion of Type Abstraction

Type hierarchy generally means that there exists a partial order for the set of types where a type at a higher position in the partial order is said to be more *generalized* than a type at a lower position in the partial order, and the latter is said to be more *specialized* than the former. However, the term *generalization* and *specialization* may be interpreted from the subsumption, composition and abstraction views.

Subsumption View of Type Hierarchy

The original view to type hierarchy taken by the object-oriented paradigm is *subsumption* [AK86]. This view is theoretically associated with *many-sorted logic*, where classes of different types are considered as different *sorts*, and the universe of discourse is regarded as comprising a relational structure in which the objects are regarded as being of various sorts. Figure 1 shows a sort lattice. We can call the most specific sort as BOTTOM which is usually interpreted as the empty sort, and call the most general sort as TOP which is usually interpreted as containing "everything" in the discourse.

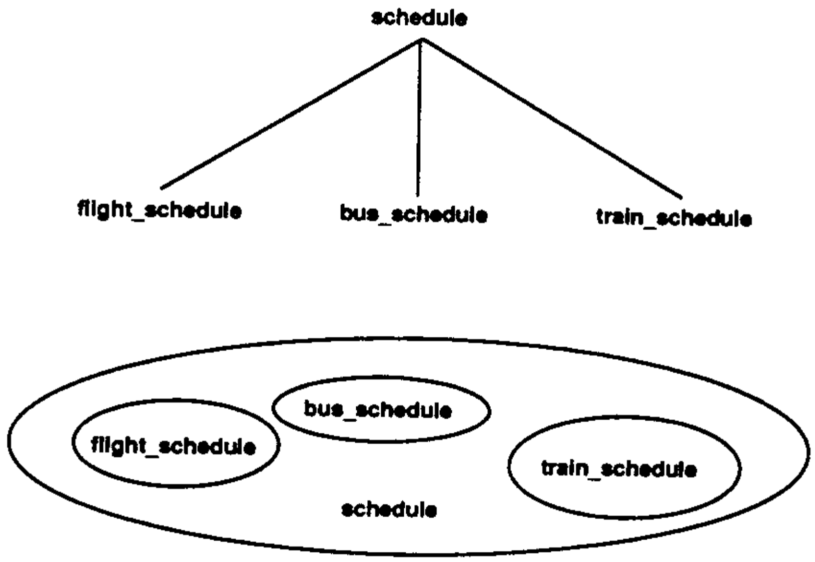


Figure 1 An Example of Type Subsumption Hierarchy.

When a sort S_1 is more general than another sort S_2, the position of S_1 is higher than the position of S_2 in the sort lattice. In this case, S_1 is referred to as a *super-sort* of S_2, and S_2 is referred to as a *sub-sort* of S_1, and the relationship between S_1 and S_2 is denoted as $S_2 \leq^{\circ} S_1$. The $\leq^{\circ}$ relationship is a *partial order*, that is, $\leq^{\circ}$ is reflexive, transitive and antisymmetric, thus forming a partial order.

Because of the synonym of **sort names** and **type symbols**, the notion of concept subsumption indeed captures the semantic relationship among *types*. However, due to its lack of revealing structural differences and constraints among objects, subsumption cannot be used for expressing the structural relationships among objects of various types.

Composition View of Type Hierarchy

The handling of complex objects in object-oriented databases introduces the *composition* relationship among types, where a type is composed of other more primitive types. We should distinguish the multilevel type configuration from the type hierarchy mentioned above. However, type hierarchy can also be considered from composition point of view, in the sense that a generalized type has the *structure* commonly contained in the specialized types.

This view can also be formalized by the lattice theory under the **structural sub-type relationship** as described below. The concept that the super-type structure is a *generalization* of the corresponding sub-type structure, and the sub-type structure is a *specialization* of the super-type structure means that the super-type structure is somehow "contained" in its sub-type structures. For example, tuple types *(name, phone#, school)* and *(name, phone#, company)* are sub-type structures of *(name, phone#)*. If we assign a type name *EMPLOYEE* to the former and *PERSON* to the latter, we can clearly see that *EMPLOYEE* is a sub-type of *PERSON*.

Considering atomic, tuple and set type structures, when we say type T_1 is the *sub-type structure* of T_2, we mean one of the following:

a) T_1 and T_2 are the same atomic types.
For example, type COLOR is a sub-type structure of itself.

b) T_1 and T_2 are *tuples* and every component of T_2 is the sub-type structure of the corresponding component of T_1 on the same attribute.
For example, (Name, Sex, School, Grade) is a sub-type structure of (Name, Sex).

c) T_1 and T_2 are *sets* and the member type of T_2 is a sub-type structure of the member type of T_1.
For example, {(Name, Sex, School, Grade)} is a sub-type structure of {(Name, Sex)}.

Note that we are **not** discussing from the usual PART_OF relationship point of view, but the generalization notion. In fact, the *structural sub-type relationship* is just the inverse of the structural *sub-object relationship* described in [BANC86]. Under the structural sub-type relationship, a super-type has a simpler structure than its sub-type. However, under the structural sub-object relationship, a super object has a more complex structure than its sub-objects. The set of all type structures under the structural sub-type relationship forms a partial order lattice, which is the *dual-lattice* of the lattice formed under sub-object relationship. We denote the fact that T is a sub-type structure of T' as $T \leq^* T'$ as shown by the example illustrated in Figure 2.

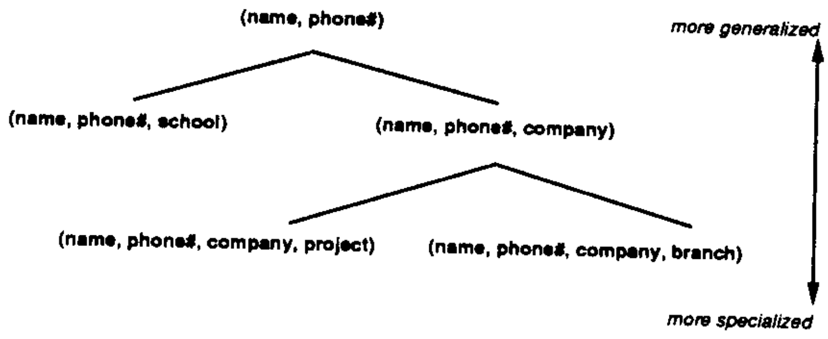


Figure 2 An Example of Type Composition Hierarchy.

The notion of composition of types thus captures the structural relationships among types. In fact, the *common structure* of a type over its sub-types is the intersection of the structures of its sub-types under the $\leq^*$ relationship. This notion is also not semantically rich enough for representing a complete semantic link between various types. On one hand, it cannot represent the semantics of the concept subsumption. For example, given the following two types:

communication(name, address).

phone(name, phone#).

from the subsumption point of view, "communication" is the super-type of "phone". Structurally they are not compatible since they contain different component types (i.e., attributes names: "address" and "phone#").

On the other hand, it does not distinguish types that are structurally equivalent but conceptually different. For example, consider the following two types:

student(id, name, sex).

dog(id, name, sex).

These two types contain the same structure (id, name, sex), but represent two different concepts.

Abstraction View of Type Hierarchy

Let us now consider the *abstraction* view of type hierarchy, where a super-class is considered to convey a *more abstract representation* than its sub-classes, namely, the domain of the super type T' is abstract over the domain of the sub-type T. We denote such a fact as $T \leq^{\wedge} T'$. Under this notion, an object may be viewed either as an instance of a type T with a corresponding representation or as the instance of a more general type T' with a more abstract representation conformed by type T'. The sense of abstraction concerns not only the type intension, but also the instance extension, that is, the same piece of information can be represented differently at different knowledge representation levels. This view is generally absent in the current notion of type hierarchy.

Under this view, an instance may have different representations at different abstraction levels. For example, let us consider two atomic types "area" and "airport", where the former is the super-type of the latter. To indicate the location of BURBANK airport, we can either use the instance of type "airport" expressed as

airport(BURBANK)

at the relatively lower knowledge level, or alternatively use the instance of type "area" expressed as

area(Los_Angeles)

at the relatively lower knowledge level. In other words, there exist mappings between the instance values from a sub-class domain and the corresponding abstract instance values from its super-class domain. For example, the following mapping exists between type "airport" and type "area" in the above example:

airport:BURBANK $\rightarrow$ area:Los_Angeles

as shown in Figure 3.

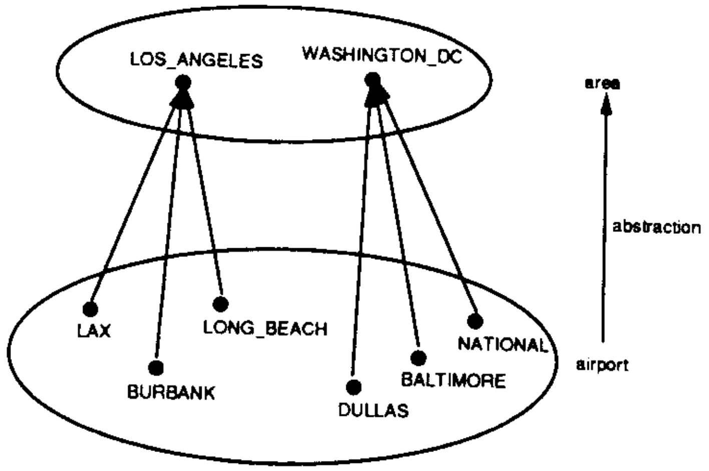


Figure 3 An Example of Type Abstraction.

Figure 4 shows another example where the instances of type "fare" for traveling by airplane, bus or train can have more abstract representations at a higher level as the instances of types "flight_cost", "bus_cost" and "train_cost" respectively. Going one level further, under type "cost" they can have even general representations.

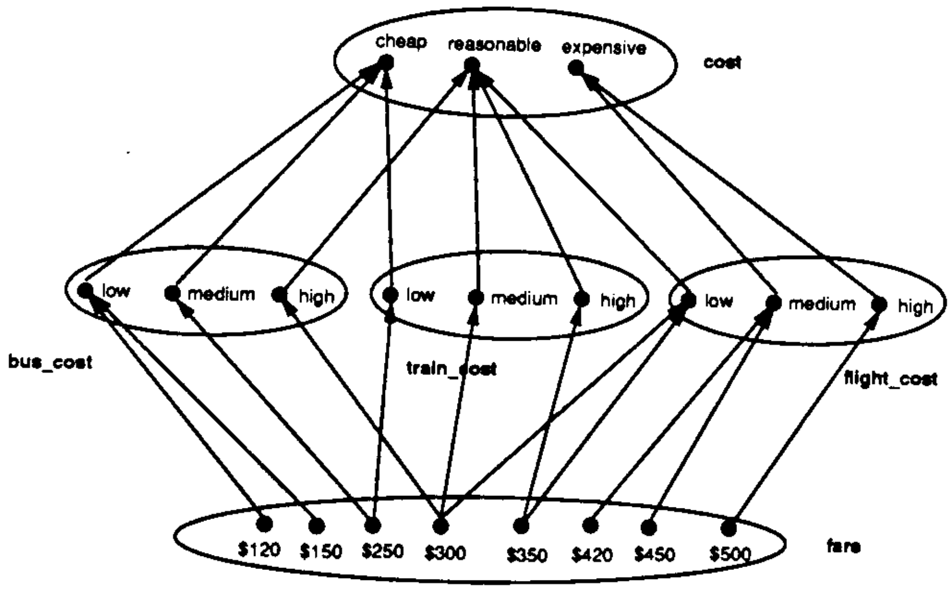


Figure 4 A Multilevel Type Abstraction Example.

This feature characterizes the notion of *type abstraction* very well. The knowledge about different object representations at different levels is essential for transferring inference between different knowledge levels.

of type "CC_JOURNEY", "CC_FLIGHT" and "DELTA_FLIGHT" are formally stated as follows :

CC_JOURNEY{**cc_journey**(departure_area, arrival_area, duration, cost)}.
CC_FLIGHT{**cc_flight**(departure_area, arrival_area, departure_period, arrival_period, hours, flight_cost)}.
DELTA_FLIGHT{**DELTA_flight**(flight#, departure_airport, arrival_airport, departure_time, arrival_time, hours, fare)}.

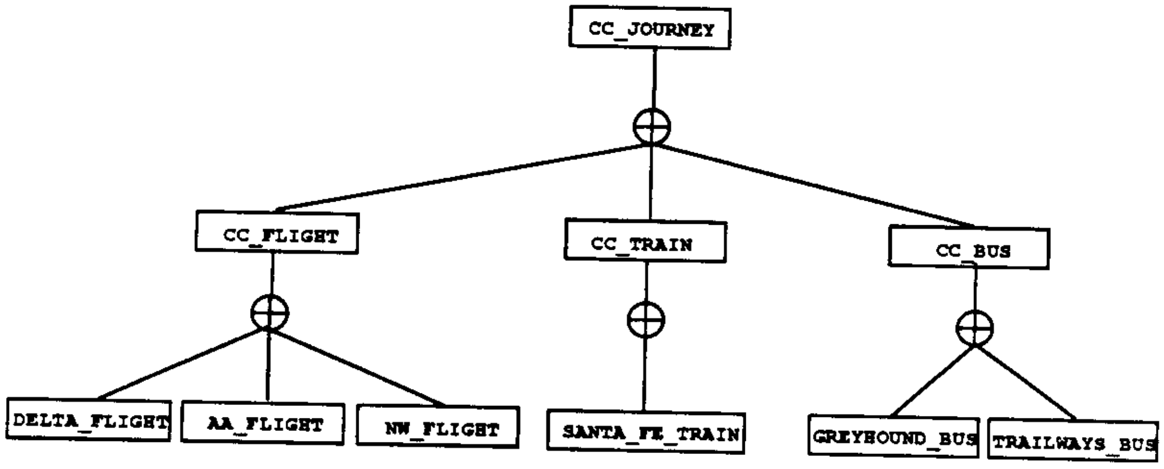


Figure 5 A Type Abstraction Hierarchy.

Now let us demonstrate step by step the generalized subtyping relationship between type "DELTA_FLIGHT" and "CC_FLIGHT".

Since

departure_airport $\leq^{\wedge}$ departure_area,
arrival_airport $\leq^{\wedge}$ arrival_area,
departure_time $\leq^{\wedge}$ departure_period,
arrival_time $\leq^{\wedge}$ arrival_period,
hours $\leq^{\circ}$ hours,
fare $\leq^{\wedge}$ flight_cost,

We have

3. Type Abstraction Hierarchy : The Integrated View of Type Hierarchy

The introduction of the type abstraction together with the type subsumption and type composition, form an integrated view of type hierarchy. In our proposed notion of **Integrated Type Hierarchy**, to capture the notion of *subsumption*, type names are used to conceptually subsume various sorts of objects, where Unique Name Assumption (UNA) on types is adopted. To capture the notion of *composition*, the structural sub-type relationship is adopted. To capture the notion of *abstraction*, different representations of an object at different knowledge levels is considered. Denoting the integrated subtyping relationship as $\leq$, for types T and T', the expression $T \leq T'$ means :

> the domain of T' is abstract over the domain of T (i.e. $T \leq^{\wedge}$ T'), or under the special case, the domain of T' subsumes the domain of T (i.e. $T \leq^{\circ}$ T'), and
> the type structure of T is contained in the type structure of T' (i.e. $T \leq^{*}$ T').

It can be seen that *subsumption is a special case of abstraction.* Thus we shall refer to the type hierarchy underlain by the above integrated subtyping relationship as **type abstraction hierarchy**.

More formally we can consider a type as an atomic type, or as a tuple-type T:(T_1,..., T_n), where T_1,..., T_n are types, or as a set-type, T:{S}, where S is a type. The domain of type T is denoted as $dom(T)$. Then based on the type abstraction notion, a type T is a sub-type of a type T', denoted as $T \leq T'$, is defined recursively as:

a) For atomic types T and T', $T \leq T'$ iff there exists a mapping **abst** : $dom(T)$ --> $dom(T')$.

b) For tuple-types T:(T_1,..,T_n) and T':(T_1',..,Tm') where n $\geq$ m, $T \leq T'$ iff $\forall i \in \{1,..,m\}$ $T_i \leq T_i'$.

c) For set-types T:{S} and T':{S'}, $T \leq T'$ iff $S \leq S'$.

For a given class, based on different views there may be multiple type hierarchies.

As an example, Figure 5 shows a type abstraction hierarchy where class "CC_JOURNEY" (CC stands for Cost-to-Cost) is partitioned into three sub-classes: "CC_FLIGHT", "CC_TRAIN" and "CC_BUS" whose elements are of types "cc_flight", "cc_train" and "cc_bus". These types are further refined to more specialized types. The schemes

departure_airport ≤ departure_area,
arrival_airport ≤ arrival_area,
departure_time ≤ departure_period,
arrival_time ≤ arrival_period,
hours ≤ hours,
fare ≤ flight_cost,

and then

DELTA_flight(flight#, departure_airport, arrival_airport, departure_time, arrival_time, hours, fare) ≤
cc_flight(departure_area, arrival_area, departure_period, arrival_period, hours, flight_cost).

which implies

DELTA_FLIGHT{
DELTA_flight(flight#, departure_airport, arrival_airport, departure_time, arrival_time, hours, fare)} ≤
CC_FLIGHT{
cc_flight(departure_area, arrival_area, departure_period, arrival_period, hours, flight_cost)}.

Figure 6 gives an example showing an instance of type "DELTA_flight" which is a more abstract representation at the level of "cc_flight".

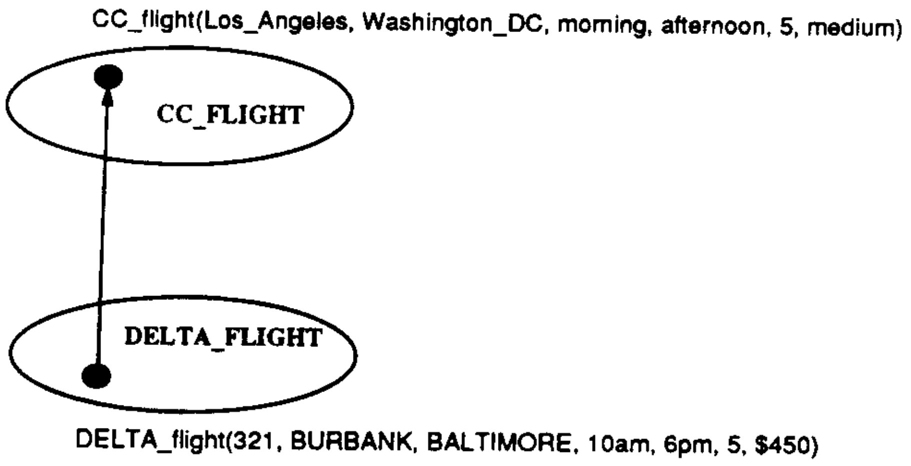

CC_flight(Los_Angeles, Washington_DC, morning, afternoon, 5, medium)
CC_FLIGHT
DELTA_FLIGHT
DELTA_flight(321, BURBANK, BALTIMORE, 10am, 6pm, 5, $450)

Figure 6 Abstract Representation of An instance.

Although the subsumption based type hierarchy is a special case of the abstraction based type hierarchy, the latter is suitable for providing links among those objects which have the same representation at a higher knowledge level. Figure 7 gives a comparison between Abstraction and Subsumption based Type Hierarchies.

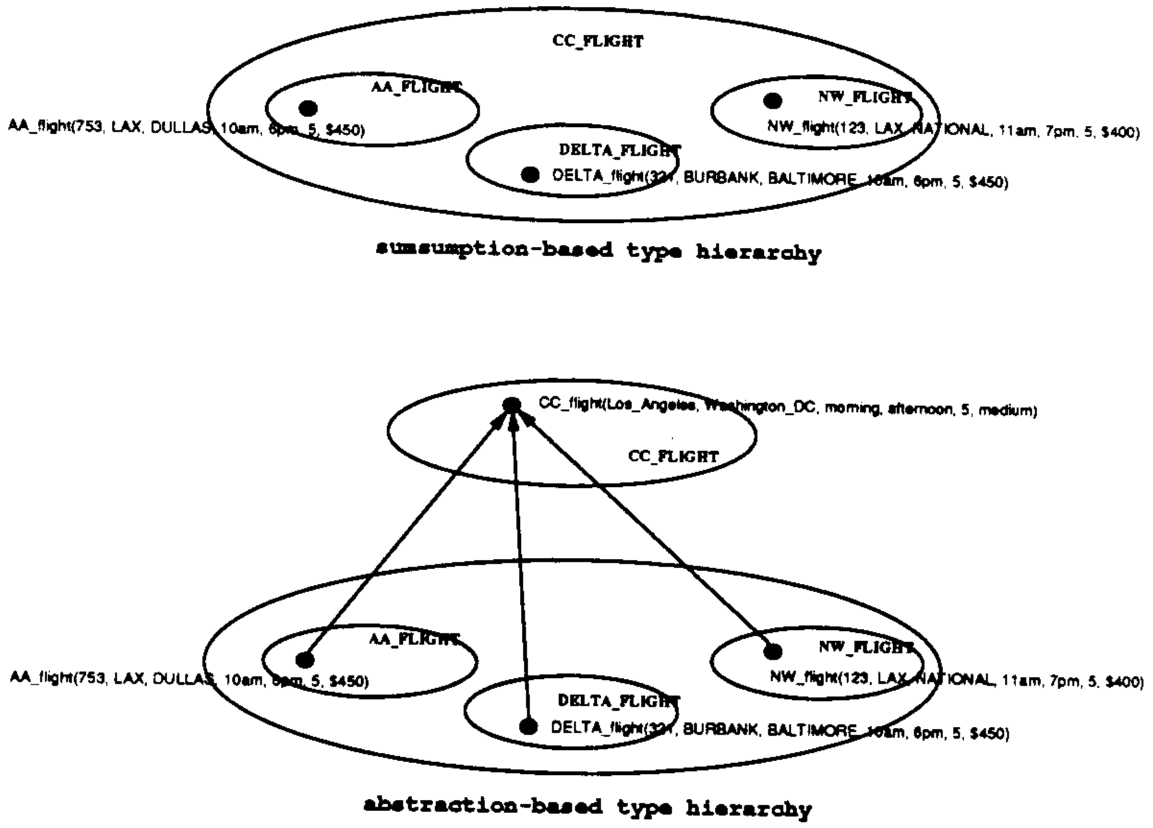

CC_FLIGHT
AA_FLIGHT
AA_flight(753, LAX, DULLAS, 10am, 6pm, 5, $450)
NW_FLIGHT
NW_flight(123, LAX, NATIONAL, 11am, 7pm, 5, $400)
DELTA_FLIGHT
DELTA_flight(321, BURBANK, BALTIMORE, 10am, 6pm, 5, $450)
sumsumption-based type hierarchy
CC_flight(Los_Angeles, Washington_DC, morning, afternoon, 5, medium)
CC_FLIGHT
AA_FLIGHT
AA_flight(753, LAX, DULLAS, 10am, 6pm, 5, $450)
NW_FLIGHT
NW_flight(123, LAX, NATIONAL, 11am, 7pm, 5, $400)
DELTA_FLIGHT
DELTA_flight(321, BURBANK, BALTIMORE, 10am, 6pm, 5, $450)
abstraction-based type hierarchy

Figure 7 A Comparison between Abstraction and Subsumption based Type Hierarchies

To summarize, we have combined three relationships in our integrated type hierarchy : subsumption, composition and abstraction. The introduction of type abstraction offers the feature of enlarging or shrinking variable scopes by moving up or down along the type abstraction hierarchy. It is easy to understand that any concept described at a higher level has larger coverage than at a lower level. A class "CC_FLIGHT" only covers the cost-to-cost flight information, its super-class "CC_JOURNEY" not only covers the information about flight, but also covers the information about bus and train. Variables that appear at different levels may thus have different coverages. This feature is very useful for cooperative query answering as will be discussed in the next section.

4. Type Abstraction Based Goal Reformulation

One of the important characteristics of type abstraction hierarchy is that inference can be performed at and between different abstraction levels. An important requirement of the decision making and problem solving process is the ability to communicate and associate between different knowledge levels for deriving conclusions. The transferring between different knowledge levels is achieved by query generalization and specialization rewriting. Processing queries at different knowledge levels consists of converting object types, attribute names, and domain values involved in the query between different knowledge levels. These are all based on the mechanisms of *type rewrite* and *term rewrite*. By *term* we mean both instances and variables of types. There are two kinds of rewrites for types and terms: *generalization rewrite* and *specialization rewrite.*

The type generalization rewrite converts a type T to its super-type T', where T' is a more abstract representation and has wider coverage than T on the type abstraction hierarchy.

The term generalization rewrite is to convert a term t, to a more abstract term t'. Assuming the existence of a distinct set of values for each type and the relations between values of atomic types, we can define the term rewrite based on type rewrite and the relations of values between different knowledge abstraction levels. The rewrite operation is based on two functions: *gen* and *sp* which represent the mappings between the values of atomic types and their subtypes. For example, the mappings

Los_Angeles = *gen*(BURBANK)
BURBANK = *sp*(Los_Angeles)

show the relations between the area of the particular airport.

The notions of type specialization rewrite and term specialization rewrite can also be defined similarly. For a type abstraction hierarchy, the specialization rewrite from an abstract type or term may yield a set of refined types or terms, which provides refined information for searching the cooperative query answers.

Query rewrite is a technique for providing cooperative query answering which consists of two processes: *Query Abstraction* and *Query Refinement*. These processes are based on the above-mentioned type and term rewrite mechanisms. In order to implement query abstraction and refinement, we need to maintain the basic knowledge such as the information about the type abstraction hierarchy and the mappings between different representations of each pair of super-type and sub-type.

The *Query Abstraction Process* converts a query Q to a more abstract query representation Q'. The process consists of the following steps :

1. Find the appropriate super-type object through type generalization rewrite.
2. Convert the attributes to the corresponding types and vice versa (since attributes associated with the same type may be separately named). This may be necessary for objects at any level.
3. Transform attributes referred to in the query to those of the super-type object through type generalization rewrite.
4. Transform conditions referred to in the query to those related to the super-type object through both type generalization rewrite and term generalization rewrite.

The *Query Refinement Process* converts a query Q to more specific query representations Q1, Q2, ..., Qn according to the type abstraction hierarchy. The process consists of the following steps :

1. Find the set of sub-type objects through type specialization rewrites.
2. Provide conversions between the attributes and the corresponding types.
3. Based on each sub-type, use type specialization rewrite to transform attributes referred to in the query to those in the sub-type object.
4. Based on each sub-type, use both type specialization rewrite and term specialization rewrite to transform conditions referred to in the query to those related to the sub-type object.

The query modification, either upward or downward, is invoked recursively depending on the requirement and knowledge availability.

5. Applications for Supporting Cooperative Query Answering

The type abstraction hierarchy can be used to support cooperative query answering in a systematic way, as will be shown in the following examples.

(a) Supporting Neighborhood Query Answering

Neighborhood inference is a type of uncertain data inference (or inexact reasoning) which may not provide the exact answers expected by the user, but still contains information which may be helpful for the user. For example, based on the type abstraction hierarchy given in Figure 5, if a user tries to reserve a DELTA airline flight from LAX airport in the Los Angeles area to NATIONAL airport in the Washington area but it is unavailable, alternative similar flights of DELTA airlines or other airlines may be given.

Neighborhood inference requires the transition of reasoning up and down to reach the neighboring objects. Given a query Q, the general processing steps are

1. search the exact type of objects required by the query Q. If failed, then
2. move upward along the type abstraction hierarchy and rewrite the query to a more general one (generalization rewrite), i.e. $Q \rightarrow Q'$. Then,
3. move downward along the hierarchy and rewrite the query to a more specific query (specialization rewrite), i.e., $Q' \rightarrow Q''$.

Three kinds of tables are utilized for assisting neighborhood inference. They are the *type_hierarchy table* describing the subtyping relationship among types, the *attribute_type table* giving the relationship between attribute names and type names which allows attributes drawn from a same type to be differently named, and the *abstract mapping tables* showing the mappings of instances each pair of super-type and sub-type, or the corresponding instance values between a super-type and a range of sub-type (e.g. "morning" corresponds to "7am to 11am"). These tables describe knowledge representations at different knowledge levels, which are stored and managed in the database. Certain query language constructs additional to SQL are introduced, such as WITHIN used to indicate set membership, and BETWEEN used to indicate a range with an upper bound and a lower bound.

For the above example, the neighborhood inference system first undergoes a *generalization* process from type **DELTA_flight** to its super type **cc_flight** (Cost-to-Cost Flight), followed by a *specialization* process from type **cc_flight** to its subtype **AA_flight**, **NW_flight** as well as **DELTA_flight** itself, to convey to the user wider ranged options, as shown in Figure 8. If all the subtypes of **cc_flight** are not available, a further generalization will be made to go up one more level to type **cc_journey** (Cost-to Cost Journey) and then some specialization step is performed to go down to **cc_train** and **cc_bus**, and to its subtypes, if any.

```
SELECT * FROM DELTA_flight
WHERE departure_airport = "LAX" AND arrival_airport = "NATIONAL" AND
        departure_time BETWEEN <9am, 10am>
```

If the above query has no answer, the system can make the following efforts to respond to the query:

1. *Generalization Rewrite* : This step is to move upward along the type hierarchy to reach **cc_flight**. Due to the generalization rewrite process, the departure_airport "LAX" is replaced by the more abstract value "Los_Angeles" on departure_area, the arrival_airport "NATIONAL" is replaced by the more abstract value "Washington_DC" on arrival_area, and the range of departure_time from 9am to 10am is generalized to the departure_period of "morning". Therefore, the query is generalized to

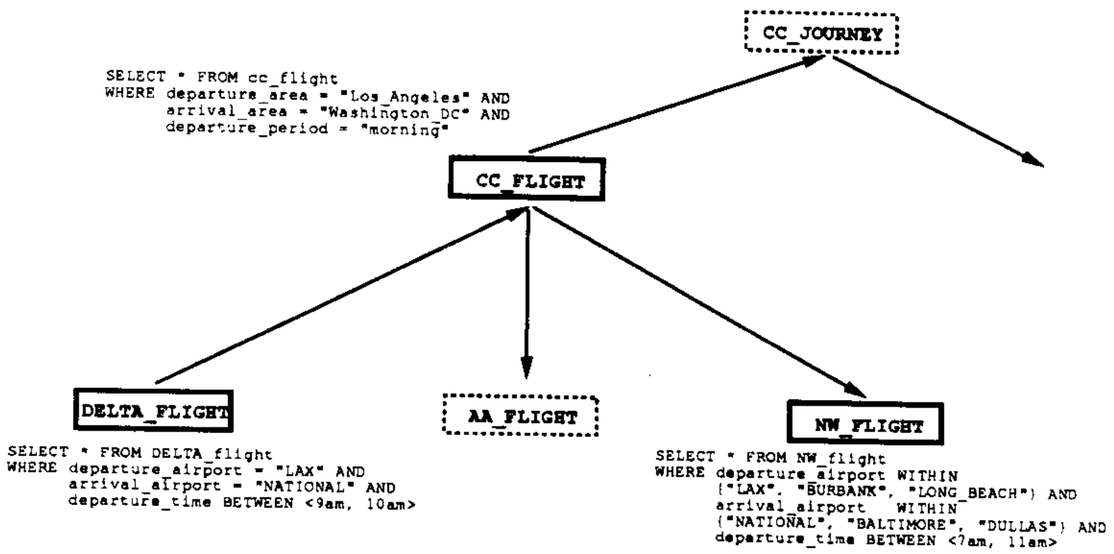
```
SELECT * FROM cc_flight
WHERE departure_area = "Los_Angeles" AND arrival_area = "Washington_DC" AND
        departure_period = "morning"
```

2. *Specialization Rewrite* : This step is to move downward along the type hierarchy to get the *neighborhood objects* of types **AA_flight**, **NW_flight** as well as **DELTA_flight** with the query specialized to the following, since all these meet the general conditions.

```
SELECT * FROM DELTA_flight
WHERE departure_airport WITHIN {"LAX", "BURBANK", "LONG_BEACH"} AND
        arrival_airport  WITHIN {"NATIONAL", "BALTIMORE", "DULLAS"} AND
        departure_time BETWEEN <7am, 11am>

SELECT * FROM AA_flight
WHERE departure_airport WITHIN {"LAX", "BURBANK", "LONG_BEACH"} AND
        arrival_airport  WITHIN {"NATIONAL", "BALTIMORE", "DULLAS"} AND
        departure_time BETWEEN <7am, 11am>

SELECT * FROM NW_flight
WHERE departure_airport WITHIN {"LAX", "BURBANK", "LONG_BEACH"} AND
        arrival_airport   WITHIN {"NATIONAL", "BALTIMORE", "DULLAS"} AND
        departure_time BETWEEN <7am, 11am>
```



```
SELECT * FROM cc_flight
WHERE departure_area = "Los_Angeles" AND
      arrival_area = "Washington_DC" AND
      departure_period = "morning"
```

```
SELECT * FROM DELTA_flight
WHERE departure_airport = "LAX" AND
      arrival_airport = "NATIONAL" AND
      departure_time BETWEEN <9am, 10am>
```

```
SELECT * FROM NW_flight
WHERE departure_airport WITHIN
      {"LAX", "BURBANK", "LONG_BEACH"} AND
      arrival_airport   WITHIN
      {"NATIONAL", "BALTIMORE", "DULLAS"} AND
      departure_time BETWEEN <7am, 11am>
```

Figure 8. Transiting a Query to the Neighborhood Objects

(b) Supporting Conceptual Query Answering

Very often a user has some question in mind but does not know *exactly* how to formulate a query. For example, if a user wants to get information about traveling from Los Angeles to Washington D.C., but is unfamiliar with particular airlines, buses or trains, he does not even need to focus on any particular transportation facilities, since information about any facility may be helpful.

The proposed type abstraction mechanism is also suitable for answering conceptual queries. Often a user does not know the best way to express a query. With this approach, the user may ask a question in a more general way. As shown in Figure 9, to get the information about traveling from Los Angeles to Washington D.C. at a reasonable cost, the following conceptual query can be asked :

```
SELECT * FROM cc_journey
WHERE departure_area = "Los_Angeles" AND arrival_area = "Washington_DC" AND
         cost = "reasonable"
```

With the *abstract type hierarchy*, such a query can be automatically refined to the following more detailed queries:

```
SELECT * FROM cc_flight
WHERE departure_area = "Los_Angeles" AND arrival_area = "Washington_DC" AND
         flight_cost = "low"

SELECT * FROM cc_train
WHERE departure_area = "Los_Angeles" AND arrival_area = "Washington_DC" AND
         train_cost WITHIN {"high", "medium"}

SELECT * FROM cc_bus
WHERE departure_area = "Los_Angeles" AND arrival_area = "Washington_DC" AND
         bus_cost = "high"
```

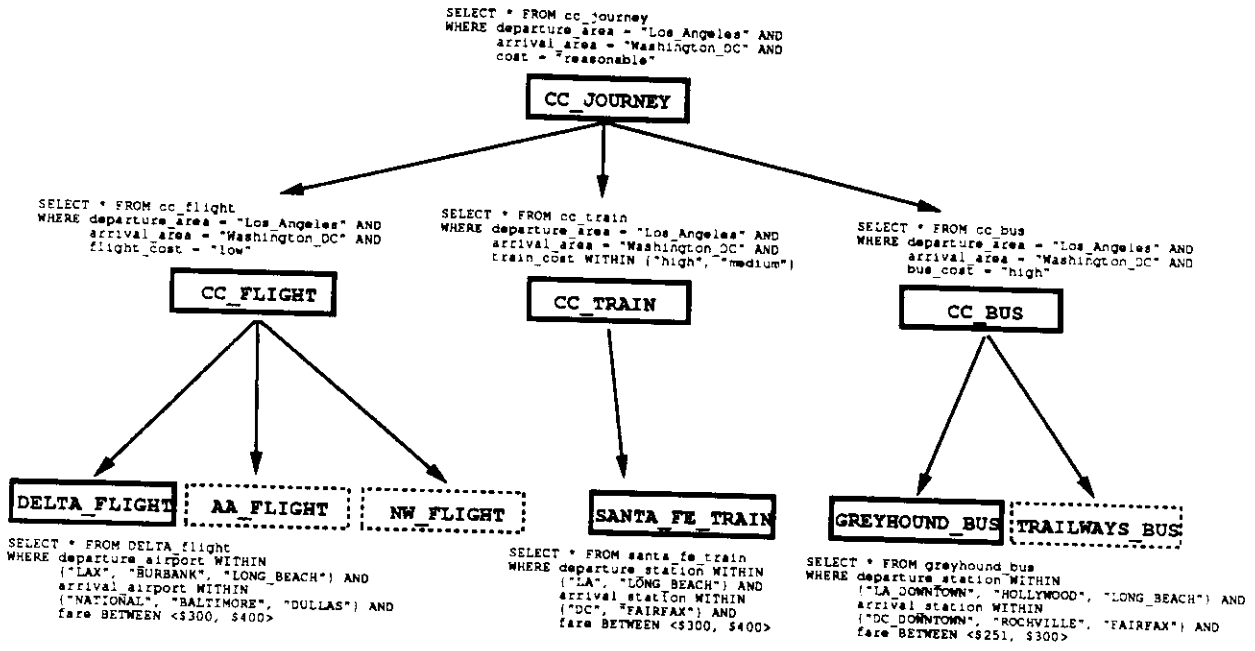
Note that in the detailed queries, **cc_journey** is "refined" to **cc_flight**, **cc_bus** and **cc_train**. Then these three queries can be further refined 1 level down, as

```
SELECT * FROM DELTA_flight
WHERE departure_airport WITHIN {"LAX", "BURBANK", "LONG_BEACH"} AND
         arrival_airport WITHIN {"NATIONAL", "BALTIMORE", "DULLAS"} AND
         fare BETWEEN <300, 400>

SELECT * FROM santa_fe_train
WHERE departure_station WITHIN {"LA", "LONG_BEACH"} AND
         arrival_station WITHIN {"DC", "FAIRFAX"} AND
         fare BETWEEN <300, 400>

SELECT * FROM greyhound_bus
WHERE departure_station WITHIN {"LA", "HOLLYWOOD", "LONG_BEACH"} AND
         arrival_station WITHIN {"DC_DOWNTOWN", "ROCHVILLE", "FAIRFAX"} AND
         fare BETWEEN <251, 300>
```

and so on.



```
SELECT * FROM cc_journey
WHERE departure_area = "Los_Angeles" AND
      arrival_area = "Washington_DC" AND
      cost = "reasonable"
```

```
SELECT * FROM cc_flight
WHERE departure_area = "Los_Angeles" AND
      arrival_area = "Washington_DC" AND
      flight_cost = "low"
```

```
SELECT * FROM cc_train
WHERE departure_area = "Los_Angeles" AND
      arrival_area = "Washington_DC" AND
      train_cost WITHIN ("high", "medium")
```

```
SELECT * FROM cc_bus
WHERE departure_area = "Los_Angeles" AND
      arrival_area = "Washington_DC" AND
      bus_cost = "high"
```

```
SELECT * FROM DELTA_flight
WHERE departure_airport WITHIN
      ("LAX", "BURBANK", "LONG_BEACH") AND
      arrival_airport WITHIN
      ("NATIONAL", "BALTIMORE", "DULLAS") AND
      fare BETWEEN <$300, $400>
```

```
SELECT * FROM santa_fe_train
WHERE departure_station WITHIN
      ("LA", "LONG_BEACH") AND
      arrival_station WITHIN
      ("DC", "FAIRFAX") AND
      fare BETWEEN <$300, $400>
```

```
SELECT * FROM greyhound_bus
WHERE departure_station WITHIN
      ("LA_DOWNTOWN", "HOLLYWOOD", "LONG_BEACH") AND
      arrival_station WITHIN
      ("DC_DOWNTOWN", "ROCHVILLE", "FAIRFAX") AND
      fare BETWEEN <$251, $300>
```

Figure 9 Refining a Conceptual Query

Thus a cooperative query answering system can provide users with the capability of processing *conceptual queries*. Based on the hierarchy of knowledge, a conceptual query can be processed to derive a set of more specific queries which can then be answered by the conventional query processing system.

(c) Supporting Associative Query Answering

Problem solving usually requires the association of various subjects. In general association between related types of objects can be expressed by rules, therefore we can view these rules as links between these types. However, to express rules at a higher conceptual level is more clear and efficient than at a lower level. Furthermore, those links may be located at any abstract level, and in many cases the links between types are not located within a single type hierarchy but span multiple type hierarchies. The proposed type abstraction notion can be used to represent and handle associations between related subjects.

As a simple example shown in Figure 10, a **remote_retrieval** problem "lost_data" has no direct links to the researchers or programmers in the related fields. Instead, some general association rules can be given. At a higher level, the rule :

If system_fault is "distributed_database_malfunction",
then call System_Development tech_staff whose area is "distributed_system".

can be used to associate **sys_fault** and **tech_staff**. At a lower level, the rule :

If distribution_error is "distribute_I/O_error",
then call the programmer whose job is "network_programming".

to associate **distribution_error** and **programmer**. It is this fact that to give such general rules is more reasonable than to give a number of lower level and tedious rules.

Association supported by the proposed type abstraction hierarchy has three typical processes :

(a) *abstracting* the problem,
(b) *leaping* related type hierarchy, and
(c) *refining* the solution.

The abstraction process is via type generalization rewrite and term generalization rewrite. In the above example, to reach the level where association "maintenance" can be referred, it goes from **remote_retrieval** to **distribution_error**, and then to **sys_fault**, accompanied with the attribute value rewritten from "lost_data" to "distributed_I/O_error", and then to "distributed_database_malfunction".

Related type hierarchies are linked via rules. The issue is that such a link only exists at a certain level. That is, the rules only use object representations given at a certain level. In this example, at the **sys_fault** level, the inference leaps to the node of another hierarchy **tech_staff** based on the rule given above, where the condition "sys_fault is distributed_database_malfunction" is turned to "area is distributed_system".

The refinement process is via type specialization rewrite and term specialization rewrite. In this example, the refinement process leads to the solution "researchers whose field is database or network", or "programmers whose job is network_programming".

These processes are carried out automatically by the system. The user only states a request such as

ASSOCIATE maintenance WITH remote_retrieval
WHERE fault = "lost_data"

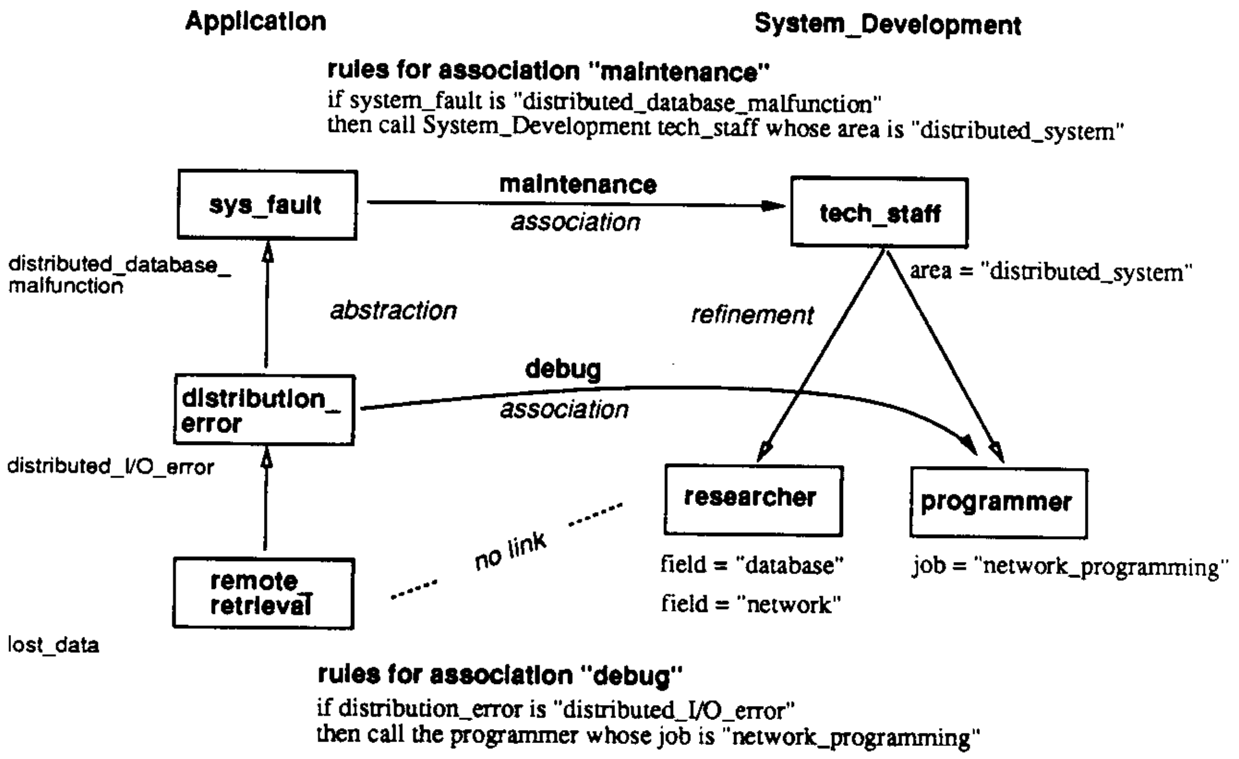

Application
System_Development
rules for association "maintenance"
if system_fault is "distributed_database_malfunction"
then call System_Development tech_staff whose area is "distributed_system"
sys_fault
maintenance
association
tech_staff
distributed_database_
malfunction
area = "distributed_system"
abstraction
refinement
debug
association
distribution_error
distributed_I/O_error
researcher
programmer
no link
remote_retrieval
field = "database"
field = "network"
job = "network_programming"
lost_data
rules for association "debug"
if distribution_error is "distributed_I/O_error"
then call the programmer whose job is "network_programming"

Figure 10. An Example for Associative Query Answering

This example indicates that the notion of association is concerned about the **leap between concepts**, and also the **levels of abstraction**. The association between these levels is the better way to capture concepts, since certain knowledge can only be found at certain conceptual levels.

In general, to support cooperative query processing through subject association, our system first moves up, if necessary, to reach a certain knowledge level where the association is defined. Then, via the association link and rules, it can transfer to another type hierarchy, and then refine the associated objects.

It can be seen that by using the type abstraction hierarchy, the rules may be specified at a higher knowledge level in terms of the object instances represented at that level. Considering the rule used in the above example

> If system_fault is "distributed_database_malfunction",
> then call System_Development tech_staff whose area is "distributed_system".

Note that the instance "distributed_database_malfunction" referred in this rule is more abstract than the more detailed instances such as "lost_data", "transmission_error", "disconnection", ...etc. Thus using this rule is more general and convenient than using a number of rules referring "lost_data", "transmission_error", "disconnection", ...etc. As a result, the number of rules in the knowledge base system can be considerably reduced and thus the inference processes can be greatly simplified. Organizing knowledge based on the type abstraction hierarchy provides a systematic way to perform inference. Furthermore, such knowledge can be organized and managed by the database system.

6. Comparison with Related Work

The concept of abstraction has received much attention. Accompanying this popularity is an extensive use of the term in many different contexts, representing varying but related notions. The proposed notion of *type abstraction* is characterized by *multi-level object representations*, namely, describing the same object differently at different knowledge levels. As a result, reasoning can be performed between different knowledge levels to derive information with various degrees of abstraction and coverage. Below we shall compare this notion with the conventional notions on *group, structure and function* abstractions. Particularly, we shall point out the differences between our approach and the semantic data model and object-oriented databases.

The mechanism of classification allows objects to be grouped on the basis of the common properties they share, so that these properties can be specified "abstractly" over all the members of the group, rather than against each particular individual [HAM78] [BRO81] [SMI77]. Multilevel classification groups objects by unioning the classes of disjunctive types. Following up is the notion of type subsumption underlying the generalization hierarchy of object types [SU83] [BRO84] [BRO86]. The corresponding *is_a* relationship sorts object types by their names. For an individual object, the notion of classification depicts the group range it falls into, and the notion of generalization allows the rename of its type to a supertype and the inheritance of property from that supertype. However, the different representations of it at different abstract levels are not handled by these notions.

Structure abstraction is usually expressed by the notions of aggregation [SMI77] and complex object [ZANI85] [BAN87] [CHEN88]. For example, a car can be viewed as the aggregation of wheels, engine, body, ...etc, by simply mentioning "car" all those components are suppressed. However, this kind of structural abstraction only forms one aspect of knowledge abstraction, and is very different from the type abstraction notion discussed in this paper. Although in some special cases the name of an aggregate type might have abstract sense over its components (e.g. using "situation" as the abstract expression of "weather situation", "traffic situation", "road condition"), in general composition does not depict a representative abstraction of an object over its components.

Another related concept is the well known notion of Abstract Data Type (ADT) [GOG75] [GUT77]. ADT aims at the function abstraction of data structures and the algebraic validation of software development. The sense of "abstraction" of ADT comes from the principle of reducing the amount of complexity or detail that must be considered at any one time [GUT78]. However, an implementation of an ADT is an assignment of meaning to the values and operations in terms of the values and operations of another data type or set of data types, and this notion emphasizes the *functional* abstraction of the overall behavior of the *components* of an object, rather than its *representative* abstraction. In knowledge engineering, it is this fact that to describe the relationship between different representations of an object directly by experience-based *plausible facts* is often more practical than by *algebraic axioms*, and the handling of such facts can be reasonably supported by databases.

Many of the above notions are represented in the object-oriented paradigm which has gained much popularity in the design and implementation of database systems in recent years [GOLD81] [WOEL86] [KIM89]. The philosophy of object-orientation is to *localize* but *not isolate* the handling of different types of objects through certain basic mechanisms such as *encapsulation* and *inheritance*. Under the notion of encapsulation, objects are classified so that their common behaviors can be specified. The behavior of an individual object is *encapsulated* in the operations called *methods* defined within the class, which is not visible from outside of the object. Under the notion of inheritance, classes are linked by the *IS-A* relationship according to their generality. If class A *IS-A* class B then A can *inherit* properties from B. This furthers the property sharing and information hiding capabilities gained by grouping objects into classes. These notions are jointly expressed by the specification of class hierarchy, or **type hierarchy**. However, the existing notion of type hierarchy is still weak in supporting *multi-level object representations*, that is, describing the same object differently at different knowledge levels. This omission can be explained as follows :

1. In the basic object-oriented paradigm, type hierarchy is only considered from a subsumption (*IS-A*) point of view. In the complex object modeling approach, type hierarchy is considered from a composition (*part_of*) point of view [BANC86] [BEE86] [TSU86] [ABI87] [CHEN89]. The representative abstraction of an object viewed from different knowledge levels is not covered by these notions.
2. The current object-oriented databases deal with information sited at two general layers : the *meta-layer* (intension) and the *instance layer* (extension). This fact is inherently implied by the mechanism of method inheritance. If a class of objects inherits a function

from a super-class, then it means objects in these two classes are actually at the same instance layer. For example, consider the age of a person, say 17, which is countable by a certain function. However, knowledge abstraction can represent an abstract concept of age, say *young*, at a higher layer and the concept does not have to be countable by the same function. For knowledge abstraction, there is no theoretical limitation on the number of layers for knowledge (or concept) representation. There may be another higher meta layer above a meta-layer. The knowledge represented at different layers may be *convertible*, rather than directly inheritable. Thus limiting object representation to only two layers, intension layer and extension layer, though applicable for handling data, is not sufficient for handling semantic knowledge.

Finally we would like to compare our approach to some rule based semantic association approach such as [CUP89]. It is true that using rules can provide various links between objects. However, without a clean organization principle, the database and knowledge base tend to become two separated parts of a system and the inference using both data and rules becomes inefficient. The proposed type abstraction hierarchy couples data extension and data intensional knowledge. Particularly, the mappings between different representations of objects at different abstract levels are themselves useful knowledge for cooperative query answering and can be stored and managed by database systems. Our proposed methodology provides support to cooperative query answering by the model based, systematic methodology rather than using ad-hoc rule-based reasoning.

7. Conclusions

The type abstraction hierarchy, query rewrite, and subject association are useful tools for supporting cooperative query answering in knowledge-based data processing environments. The proposed type abstraction hierarchy integrates the notions of subsumption, composition and abstraction and thus offers an integrated view of the type hierarchy with multi-level knowledge abstraction. Based on such type abstraction hierarchy, the query rewrite and subject association can be used to provide reasoning among different knowledge levels to derive cooperative answers and should play an important role in decision making and problem solving systems.

Data/knowledge base application requires complex tasks performed through the intelligent cooperation of multiple knowledge sources and agents based on the knowledge represented at various abstract levels. Since the object representative hierarchy can be reasonably handled by databases, and the type abstraction hierarchy can be coupled with knowledge processing, our proposed methodology is well suitable for such applications. We have implemented a prototype cooperative database system at UCLA by using this methodology. Our preliminary experimental results reveals that this approach provides a systematic and efficient way for cooperative query answering.

References

[ABI87] S. Abiteboul, S. Grumbach, "Une Approche Logique de la Manipulation d'objets Complexes", 3e Journees BD3, Port Camargue, 1987.

[AK86] H. Ait-Kaci, "Type Subsumption as a Model of Computation", in "Expert Database Systems", Kersburg ed. 1986.

[AL90] A. Alashqur, S. Su and H. Lam, "A Rule-based Language for Deductive Object-Oriented Databases", Proc. of the 6th International Conference on Data Engineering, 1990.

[BANC86] F. Bancilhon and S. Khoshafian, "A Calculus for Complex Objects", Proc. PODS'86.

[BAN87] J. Banerjee et al. "Data Model Issues for Object-Oriented Applications", ACM Trans. on Office Information Systems, 1987.

[BEE86] C. Beeri et.al, "Sets and Negation in a Logic Database Language LDL1", MCC Rep,1986.

[BRO81] M. Brodie, "On Modeling Behavioral Semantics of Databases", Proc. of VLDB'81, 1981.

[BRO84] M. Brodie, J. Mylopoulos, J. Shmidt eds, "On Conceptual Modeling", Springer-Verlag, NY, 1984.

[BRO86] M. Brodie, J. Mylopoulos eds, "On Knowledge Base Management Systems", Springer-Verlag, NY, 1986.

[CHEN88] Q. Chen and G. Gardarin, "An Implementation Model for Reasoning with Complex Objects", Proc. of ACM-SIGMOD 88, USA, 1988.

[CHEN89] Q. Chen & Wesley Chu, "A High Order Logic Programming Language (HILOG) for NON-1NF Deductive Databases," Proc. of 1st International Conference on Deductive and Object-Oriented Databases, 1989, Japan. (also in "Deductive and Object-Oriented Databases," Elsevier Science Publishers B.V., 1989).

[CUP89] F. Cuppens and R. Demolombe, "Cooperative Answering: A Methodology to Provide Intelligent Access to Databases", Proc. of the 2nd international conference on expert database systems, 1989.

[GOG75] J. Goguen et al, "Abstract Data Types as Initial Algebras and Correctness of Data Representations", Proc. Conf. on Computer Graphics, Pattern Recognition and Data Structure.

[GUT77] J. Guttag, "Abstract Data Types and the Development of Data Structures", CACM 20(6), 1977.

[GUT78] J. Guttag, E. Horowitz and D. Musser, "Abstract Data Types and Software Validation", CACM 21(21), 1978.

[HAM78] M. Hammer and D. Mcleod, "The Semantic Database Model", ACM Trans. Database Systems, 6(3), 1981.

[KIM89] W. Kim, "A Model of Queries for Object-Oriented Databases", Proc. of VLDB'89, 1989.

[SMI77] J. Smith & D.C.P. Smith, "Database Abstraction : Aggregation and Generalization", ACM trans. Database Systems 2(2), 1977.

[SU83] S. Su, "SAM*: A Semantic Association Model for Corporate and Scientific-Statistical Databases", Information Sciences 29, 1983.

[TSU86] S. Tsur and C. Zaniolo, "LDL: A Logic Based Data Language", Proc. VLDB 12, 1986.

[ZANI85] C. Zaniolo, "The Representation and Deductive Retrieval of Complex Objects", Proc. VLDB 11, 1985.