Discovering and Resolving User Intent in Heterogeneous Databases (Extended Version)

Chris Fernandes[1] and Lawrence Henschen[2]

[1] Department of Computer Science,
Colby College, Waterville, ME 04901
cfernand@colby.edu
[2] Department of Electrical and Computer Engineering,
Northwestern University, Evanston, IL 60201
henschen@ece.northwestern.edu

Abstract. We propose a system whereby subtle semantic ambiguity found in queries of distributed heterogeneous database systems can be resolved by considering the user's intentions. Through the use of domain-specific knowledge embedded within a mediator-based architecture, subtleties in meaning can be explicitly modeled. Through the use of dynamic profiles and active dialogue, the system can discover user intent, providing more satisfying query answers.

1 Introduction and Problem Statement

Modern heterogeneous database systems generally require users to issue queries via a global query language. This is because, typically, no one member database of the distributed system has all the concepts of the whole system. Moreover, two identical terms in two different local schemas may have slight semantic differences in the context of the entire network. Consider a network of databases dealing with colleges and universities. They all keep track of students, but the term *student* may have different meanings depending on which local database is involved. A community college will have a different idea of what a student is than a university with a graduate program. But both databases may use the same term, *student*, by which to represent them. If a query is issued to find a student roster, this difference does not come into play. However, if a query is issued to find all members of a graduate student union, the system should not even bother searching the community college database since it can never return answers. This example demonstrates the problem of *conditional attribute equivalence* – the idea that local concepts in different databases may conditionally be thought of as the same object depending on the query.

2 Globally vs. Locally-expressed Queries

To avoid the ambiguity caused by this heterogeneity, many systems have a global language containing a vocabulary of terms with exact pre-specified meanings.

Wrappers are then used to translate global terms into their meanings within a local schema's context.

This global approach places excess burden on the users to first understand the semantic differences that terms may have in different databases and then to very carefully express queries to precisely reflect their desired semantics. This disadvantage can manifest itself in three ways.

First, note that global concepts are often expressed as generalizations that are broad enough to cover the varying meanings of that concept at all the local levels. The user has to be aware of the different nuances that terms might have in the other local repositories and add appropriate constraints to be sure his/her intent is specified. In fact, it is even possible that some functionality might be lost if too much generalization occurs. For example, consider a database of domestic movie titles. If this local repository joins the Internet Movie Database (IMDB), the concept of *movie title* now includes foreign films simply due to the fact that the IMDB is a much larger repository. A user asking for movie titles in the original database is, at least implicitly, asking for domestic titles. If the IMDB has no attribute or other way to distinguish domestic titles from international ones, adding an extra condition "domestic" to a global query about movie titles is useless. The user can literally no longer ask for domestic movies only, like s/he could (implicitly) do before his/her local repository became a part of the global network. The original meaning of the local term *movie title* has been sacrificed in favor of the more generalized term present in the global query language. However, there may be times when the user wants to use the semantics associated with the local term and not those associated with the global term. As in the example above, users may not be able to ask certain queries anymore if the local terms do not have the same meaning as their global counterparts. In addition, a user may like to know which other local databases in the network match his/her own local database's semantics. This too cannot be accomplished by a global query language.

A common counterargument to this point of wanting to use local terms to express global queries is to suggest the use of a view to show the user the local database's original semantics. While much has been written about the usefulness of views (see [7] for discussion), a view is simply a virtual relation made up of already existing terms in one or more databases. It would allow the user to issue queries using familiar terms, but those terms would still have the globally-used semantics. So while the use of a view is not disadvantageous, it does not invalidate this point.

Second, as new databases join the global network, existing terms may take on even more nuances, new terms and concepts may enter the global language, and perhaps even the global schema itself may change. Users will then need to become familiar with the changes, and so user training in a dynamic network is not a one-time affair like many researchers claim.

Third, consider that many distributed networks consist of member databases that, before becoming part of the network, only issued local queries to locally held data. Many of these existing systems would already have a set of appli-

cations that use the database via its local schema. Banks use batch programs to process daily transactions, and brokerage houses use monitoring software to update investment portfolios. It would be beneficial both cost-wise and time-wise for these applications to still be usable once the database becomes part of a distributed system. But under a global language-based system, applications would need to be rewritten. And they may also need additional updating as the network changes. For the same reason, commonly run queries that were written and saved under an older incarnation of the global schema may also need to be updated.

The opposite approach allows users to specify queries in their own local database language and provides translators to map those queries to a global form that reflects the semantics of that local database. This allows users to issue distributed queries and to correctly interpret answers without having to learn a new language and be continually retrained. This would be a distinct advantage for non-sophisticated users, and it is reasonable to expect a large number of database users to fit into this category. For example, a distributed information system might include car rental companies and meteorological databases, both of which use the term "map" with related but distinct meanings. A counter agent at a car rental location may never care about seeing meteorological maps of the region, only street maps to give to customers renting cars. Such counter agents are probably not database experts and should not be required to know about all the other kinds of maps that may be available and how to specify just street maps when issuing a query. On the other hand, under the local-query approach, an advanced user who happens to know about other information at other sites and wants to take advantage of that related information must again learn the global language and continuously keep up to date as the network evolves. A travel planner in the "maps" example may very well want one or both kinds of maps when planning a group tour to account for both the actual transportation as well as possible weather-related contingencies. Even more, the planner may issue the same query, for example "get a map of the Boston area", at different times and want different kinds of maps based on what aspect of the tour is being planned at that moment. But, again, travel planners are not likely to be database experts, and it would be advantageous to develop some kind of system that would help such a user cope with a semantically diverse and evolving distributed knowledge base.

We will describe in this paper a system that attempts to merge the best features of each approach and to help users of both kinds. Queries are expressed in local database languages, and the query-processing algorithm uses the query and a global knowledge base in combination with information from an individual user profile and even user dialogue to translate the local query into a global one. Through the use of the profile and dialogue, our system tries to discover the real intent of the user query. In addition, our system attempts to help the user specify the intent when it can't be guessed or when the user has requested help. The development and use of individual user profiles and the nature of the dialogue are the primary foci of this paper. They are described in Section 4. In Section 3, we

differentiate our work from other research in this area. Section 5 presents results from preliminary experiments with our prototype system. Section 6 presents directions we would like to take in the future and Section 7 contains concluding remarks.

3 Related Work

As first proposed by Wiederhold [13], a mediator is an entity external to the local schemas that is used to help control the integration of data and concepts between local databases. However, all translations, by definition, must be kept internal to the mediator. This means that no alterations to the local databases are necessary. The local schemas maintain their autonomy, and applications need not be rewritten in order to work after the local databases join the network. Thus the mediator actually does what its name implies: it mediates or arbitrates between the local schemas using a language all its own. Furthermore, this language is not known by the local databases, nor does it need to be.

We have chosen a mediator-based system for our implementation since it allows for the two most important criteria we desire: the ability to ask queries through local member databases and the presence of a global knowledge base to perform translation on a query-by-query basis. Since intent can change over time, having this latter property is critical. Many other prototypes are also mediated systems, but most do not consider the idea of intent as we have stated it.

Most mainstream mediator-based prototypes use a single global language to represent queries. These include rule-based languages such as the one used by the HERMES system [11], description logic-based languages such as Loom used by SIMS [2], clause-based languages such as that used by Information Manifold [6], and others such as OQL used by DISCO [12]. Using these global languages provides certain advantages. HERMES, for example, is able to incorporate a degree of probability into its rules, while Information Manifold's use of logical clauses provides a robust translation from its global language into local subqueries. However, they all prevent the user from using a local language with which s/he may already be familiar, and they all prevent the use of applications which already utilize that local language.

A means of adjusting to different user intents is another aspect found in only a few prototype systems. One such system, TSIMMIS [3], is web-based and allows for a limited type of intent clarification by including hypertext in a query result. By this hypertext the user can see more general or more specific information concerning the objects involved in the query answer. This could be extended so that the user could reissue a query in which s/he specifies a more appropriate level of object detail. We propose a dialogue system which would clearly define user intent on a level not seen in these prototypes.

Another important aspect which we include in our system, but which others do not, is conditional attribute equivalence. Essentially, this term means that any two attributes which are considered to be equivalent or at least synonymous for one query may not be considered equivalent or synonymous for another. Consider

a distributed system dealing with universities, where one local database keeps track of an undergraduate-only institution while the other holds data of a university with graduates and undergraduates. Two "student" attributes in these two local databases may be considered equivalent for a query dealing with counting up all students but quite dissimilar for a query dealing solely with graduate students. Existing systems such as OBSERVER [8] may use a single ontological term to relate a priori to multiple local attributes. As a result, there is no easy way for these equivalences to change from query to query. Our approach will use constructs called annotations [4, 9] to accomplish conditional attribute equivalence. These annotations will provide for query-dependent processing, which we believe is essential for determining user intent.

4 Discovering a User's Intentions

In this section we describe our method for accommodating user intent in the context of a distributed heterogeneous database system with varying local semantics. The presentation begins with a motivating example and brief discussion of some approaches that do not accommodate user intent to contrast existing ideas with our approach. Next we briefly describe our knowledge representation scheme to show how it is feasible to represent variations and nuances in ontologies. We then describe our method for allowing users to specify their intent and indicate how our algorithm handles such intent in query processing.

It is important for the reader to appreciate why the current mediator/wrapper model cannot accomplish what our system does. Mediated systems that use wrappers do allow users to express queries in local languages; the wrappers translate such queries to the global language for distributed processing. Unfortunately, there is still only one interpretation at the global level, and, as we have described in Section 2, a single global interpretation cannot account for nuances in meaning of terms and can even prevent users from asking the same queries they could before joining the network. The wrappers themselves actually hide these nuances in meaning between the global level and the various local databases. Thus, it is not even possible for the mediator to know about the semantics of an individual local repository and, consequently, also not possible for a user of a local database to even ask what other databases in the system might have the same semantics. In the example of Section 2, a user who really wants domestic movie titles might want to inquire about other databases that could provide such titles and go to those directly instead of using the mediated network. Our system does allow for multiple interpretations of terms. Moreover, as will be shown in Section 4.2, the knowledge about interpretations is explicit in the global representation and could be accessed to discover other databases with the same meaning for a given term. Finally, our system allows users to include or exclude databases. Part of the profile system, described in Section 4.3, allows a user to explicitly include or exclude a local repository. On the other hand, a query may itself implicitly exclude certain databases from participating in answering that query. In the example of Section 2, if the IMDB has no method

Person		
Name	Phone	State
Fred	111-1111	VA
Sally	222-2222	IL
Tom	333-3333	HI

```
SELECT name, phone
FROM Person
WHERE state="IL"
```

Fig. 1. Sample query in an Employee Database

for distinguishing domestic titles, our system will simply not access titles from that database; the user doesn't need to know about this detail of IMDB or even be aware that this is happening. To be sure, there may be some titles in IMDB that are domestic, and the user will not be given these. However, unlike the mediator/wrapper model, at least the user will still be able to ask the query and get some answers back.

4.1 A Motivating Example

We first illustrate that subtle heterogeneity can creep into even the most innocuous of queries. Figure 1 shows a table from an employee database showing each worker's name, phone number, and place of residence. The accompanying query is asking for the names and phone numbers of those employees residing in IL. While the semantics for such a query would be clear for any one relational database, it becomes more complex if this is a distributed query over many employee tables. Figure 2 shows two local databases containing identically structured employee tables, but differing in their interpretation of the **phone** attribute. In DB1, the phones are all cellular phones. They do not have a permanent location, and they are probably kept with the employee most of the time. However, in DB2, the phones are regular permanent phones, located most likely at the employee's place of residence. This difference is not reflected in either of the two tables, and it produces two different query interpretations. Did the user specify "IL" because s/he intended to call there, perhaps looking for the employee's family? Or did the user intend to contact *people* based in IL, regardless of where their phones are? Returning tuples from just one database, the other, or both may not necessarily satisfy the user's intent in all cases. The dynamic discovery of intent, therefore, is a necessity.

There are several available techniques that can be used to decide upon a particular interpretation before query run-time:

1. **Make the intent explicit in the query.** In this approach, the user must precisely specify what s/he wants by clarifying the meaning of all ambiguous terms within the query. We believe this not only places an unnecessary

DB1: cellular phones

Person		
Name	Phone	State
Fred	111-1111	VA
Sally	222-2222	IL
Tom	333-3333	HI

DB2: regular phones

Empl		
Name	Regphone	Place
Sally	444-4444	IL
Carl	555-5555	IL
Beth	666-6666	VA

Fig. 2. Subtle Heterogeneity in Logically Identical Tables

burden upon the user, but it also assumes the user's knowledge about the distributed system to be very broad. In order to explicitly clarify an unclear concept in a query, the user must be knowledgeable about similar terms in different local schemas. Not only can we not expect the user to have this knowledge, but it would defeat the purpose of wanting to use one's local schema language for query expression.

2. **The originating database determines the intent.** This approach forces the query to use the same constraints as the local database from where it originated. If the phone example above worked under this philosophy, then if the query had been issued from DB1, the system would return only those tuples dealing with cellular phones, since DB1 deals solely with cellular phones. We believe that this undermines the very nature of intention, since the user no longer has a say in the matter. Unlike the previous technique, which assumed the user had a very broad knowledge of the system, this technique assumes too narrow a view. Indeed, the user may be told or may be otherwise aware of different types of phones. If a user from DB1 is visiting and using the system at DB2, s/he should not be forced into the semantics of DB2. S/he should still be able to keep his/her own ideas of what query terms mean and have the system process queries accordingly.
3. **Always use the intent having the least number of constraints.** In this approach, all tuples which could possibly be answers are returned. But this solution is not feasible in general, especially where scalability and time are factors. In an emergency situation such as a plane crash, passenger manifests need to be used to notify family members quickly. In this scenario, permanent phone numbers would be desired, not cell phone numbers. Having the system return hundreds of extraneous tuples that the user must sift through is not practical. A popular offshoot of this technique is to provide all answers conditionally, labeling the answers which are cellular phones and separating them from home phone tuples. While more feasible, this would still require a means for determining that this semantic subtlety exists. It also still places the burden on the user, albeit at the end of the query pro-

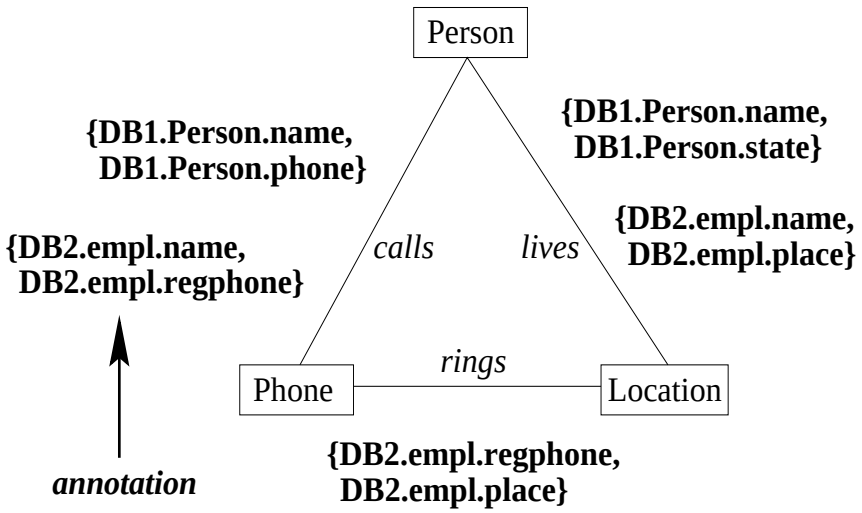

Fig. 3. Section of Mediator used for Phone Example

cessing instead of at the beginning (as in technique 1 stated above.) If time is a factor, this technique is at a distinct disadvantage.

All of these techniques take the stance that any one particular query has a unique interpretation. However, as demonstrated in the phone example, it is possible that a single query may have multiple interpretations depending on what the user wanted to do. Therefore, we do not believe that any of the above methodologies are adequate since they do not account for what the user had in mind. Our system takes the idea of interpretation one step further by allowing each *user* to have an interpretation instead of each *query* to have an interpretation.

4.2 Finding Ambiguities in Intent

In order to choose the desired interpretations of a given query, a system must be able to explicitly model subtle semantic differences like the ones given here. The ambiguity in the phone example stems from the fact that there are distinct relationships between the **phone** and **state/place** attributes in DB1 and DB2. We have developed a graph-based mediator [4, 9] that precisely and explicitly maps out concepts and relationships at the global level and relates these to the concepts and relationships in the individual local databases. Although the global representation and the related algorithms are not the focus of this paper (see [4, 9] for full discussions of these), we illustrate briefly how our system models semantic variations.

Figure 3 shows a portion of the mediator for our phone example. Nodes represent concepts that are known (in one interpretation or another) by one or more

of the member databases. Nodes are linked together by edges that represent relationships. Two nodes may have several edges between them if there are several relationships between them that are relevant to the particular domain. For example, two nodes **Car** and **Person** may be linked by two relationships: *owns* and *drives*. Both of these relationships may be true of an individual who both owns and drives an automobile, but only one may be true of an individual who is renting a car from an agency. In our system, relationships are always binary. It is also possible to utilize directed edges in the graph to indicate relationships that only apply in a certain direction. In the example above, one would not want the *owns* relationship to be interpreted as the car owning the person. A directed edge could enforce that. Since directed graphs are merely subsets of undirected graphs, the remainder of this thesis will assume that mediators are undirected and that directions can be applied as needed.

Knowledge is embedded in the mediator in the form of **annotations**. An annotation is an ordered pair $\{x, y\}$ where x and y are local database attributes. x and y are expressed using a LOREL-like notation [1] where each has the form *a.b.c.* a denotes the local repository name, b denotes the table name, and c denotes the attribute name. Each annotation is associated with a relationship R that exists between two global concepts in the mediator. The existence of annotation $\{x, y\}$ along a relationship R means that R **must** hold between local attributes x and y. Thus, the annotation $\{DB2.empl.regphone, DB2.empl.place\}$ shown in Figure 3 means that the relationship *rings* must hold between the **Phone** and **Location** concepts in local database DB2. That is, for DB2, the phone must ring in the specified place. The lack of a similar annotation for DB1 for the *rings* relationship means that *rings* does not hold or is unknown in DB1. This distinction is exactly the difference between a mobile and non-mobile phone. In general, the presence or absence of an annotation with respect to R can make explicit the subtle differences in meaning that exist between similar pairs of terms in different member databases. This is what allows the query processor to handle the problem of conditional attribute equivalence.

The reader will note that the knowledge expressed here could not be accomplished with a functional dependency. In fact, because an annotation is expressly "linked" with a specific relationship, the assertion it makes is stronger than a standard functional dependency. The dependency $Phone \rightarrow Place$ in DB2 simply means that a given phone has *exactly one* location. But another relationship (e.g. *bills-to*) would also satisfy the functional dependency even though it is possible that one relationship but not the other would be true in the same local repository. The existence of a functional dependency in one database but not the other reveals only the existence of heterogeneity, while our attachment of an annotation to a relationship discloses the nature of the heterogeneity.

The mediator in our system is created manually, most likely by the DBAs of the local systems that make up the distributed network. We agree with [5] and [10] who state that humans will still be involved at some level of the semantic integration process for some time to come. This is because there is not necessarily a "correct" graph that one should form from a given list of mediator concepts and

relationships. Since the DBAs know the nuances of their own systems best, we believe they are best suited to understand what subtle differences exist between syntactically identical terms in the member databases. Since it is critical for these subtle differences in meaning to be explicitly modeled in the mediator, DBA involvement is still necessary for an accurate representation of that knowledge. It is our hope that this process can eventually be semi-automated, especially during the process of updating the mediator structure when new local member databases are added or old ones removed.

4.3 Resolving Ambiguities in Intent

Once the source of competing interpretations has been found, the system can then move to the task of resolving the ambiguity. We have developed two methods for determining what the user truly intended: individual user profiles and on-the-fly dialogue.

A **user profile** is a list of preconceived notions that the user has about what local query concepts mean. A profile may contain domain-related information, e.g. the concept *student* means just undergraduates. It may also contain general query processing preferences, such as wanting to exclude a particular local database from contributing answers to a distributed query. If the network consisted of corporations that were competing for the same goods or services, a marketing agent may want to ask pricing queries of the system in order to check on the competition. In this instance, s/he may want to exclude his/her own database from providing answers since the originating database is not involved in what the company's competitors are doing.

A profile can be created and modified before query run-time. In the system we are developing, a profile is initially created when a new user logs into the system for the first time. At that point, the user is asked a series of questions about how s/he interprets certain local concepts. A screenshot in our system for this process is shown in Figure 4. The result is a personal knowledge base that the query processor can consult to help determine a user's intent. In our phone example, the query processor could examine this profile when it considered the *rings* relationship shown in Figure 3. If the user had indicated that only non-mobile phones are of interest, then DB1, containing only cell phones, would be eliminated from the search space.

Profiles are meant to be dynamic. After a user enters initial answers, s/he can change it at anytime while issuing queries. Initial profile settings are meant to reflect what the user *usually* assumes for most queries. However, interpretations sometimes change depending on what kind of query a particular user is issuing, so the profile can be edited for that purpose. For settings and interpretations that change frequently, the DBA has the option of including a "depends" option as a possible interpretation choice. Clicking "depends" tells the query processor that the user expects the interpretation of that term to change on a query-by-query basis. In that instance, the processor will ask the user at run-time what the interpretation should be for that query.

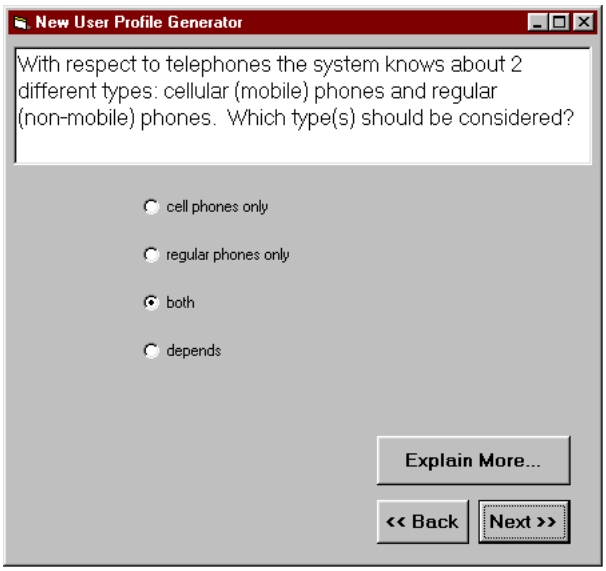

Fig. 4. Creating a User Profile

Profiles will also be changed when the system configuration changes. Adding a new member database to the network may cause a new ambiguity to arise in a particular query term (if the new member database uses that term differently than any other member database.) That will cause the DBA to add a new question to the profiler. If a change has been detected in the profiler, the system will prompt all users to give a preference the next time they login to the network. Alternatively, the DBA can set a default preference and allow users to make changes to it if they so desire. Admittedly, new ambiguities may cause the users to have to answer *all* of the profile questions again since their preference may affect other responses. The DBA will have to determine if this is the case. However, the process of updating profiles is not guaranteed to happen every time a new local database is added. It will only occur if the new member database brings new concepts to the network for which the mediator doesn't already have an interpretation. This will happen more often with a smaller network. With a large network of hundreds of local databases, it is unlikely that the addition of one more will result in a new ambiguity, especially since our system is meant to be used in an environment of a single domain such as a collection of school systems or a network of investment companies. Coupled with the fact that profiling is all done before query run-time, the process of updating profiles is not time intensive.

Profiles need not exist only for individual users, however. Profiles for specific classes of individuals could also be created. For example, all incoming business school students at a university or all new secretaries in a corporation could have a class profile that would ensure a uniform query vocabulary. The fact that a

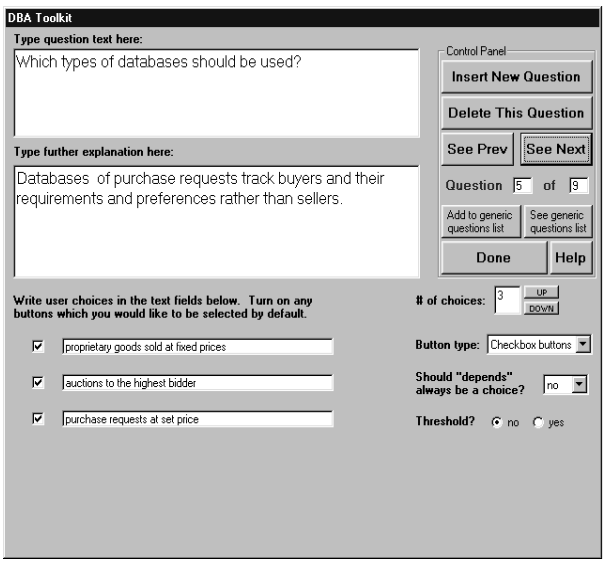

Fig. 5. DBA Toolkit for Creating Profile Generators

local query language is being used would provide a foundational interpretation of terms for creating such a class profile. Similarly, a profile could be set up for an application, extending its access from the original database to a distributed federation of databases. Moreover, as the global schema changes in response to the addition or deletion of member databases, administrators could modify these class profiles to keep end users and applications up-to-date. These changes would be transparent to non-sophisticated users while still allowing more advanced users to tweak their own profiles if they so desire.

In addition to a profile generator, we have also implemented a DBA toolkit by which profile generators can be created quickly. This toolkit is illustrated in Figure 5. We envision that both local and global database administrators are in the best position for understanding the subtle semantic differences between the syntactically identical concepts across individual member databases. The administrators can then utilize this tool to form the questions which will make up profile generators. These generators can be tailored by each local DBA so that questions are best expressed to match the level of user expertise and "lingo" in use at that local site.

By itself, a query-independent mechanism like a user profile is not sufficient. A user may be aware that his/her interpretation of a term may change over time. In a profile, this can be represented by the "depends" choice as shown in Figure 4. There may also be interpretive differences that may not be represented in a profile. In a large system where a mediator may contain hundreds of concepts, there may be many terms that may contribute to query ambiguity even though

they are used in very few queries. For those terms, it may not be worth the time it would take to ask the user about it in a profile. Instead, the system could ask the user for a query-specific interpretation on a need-to-know basis. This is the idea behind **on-the-fly** dialogue. On-the-fly dialogue asks the user to pick a particular interpretation of a term *during* query run-time. Responses the user gives affects which interpretations the system considers valid. In our phone example, the mediator would need to decide whether or not to include the *rings* relationship in the path connecting the three relevant mediator concepts. It would first consult the user's profile to see if a determination could be reached from it. If an interpretation is not given (e.g. if the user chose "depends") then the query processor would initiate on-the-fly dialogue by asking the user to provide an interpretation at that point for that query. As in the profiler, on-the-fly dialogue takes the form of questions along with a list of possible interpretations from which the user chooses. If the user truly did not have a preference as to interpretation, the processor could return two sets of answers, one for each valid path representing each interpretation.

On-the-fly dialogue is also initiated whenever ambiguity is detected by the mediator during the path finding process. Annotations are one way that this detection is made. Once the set of relevant mediator concepts is obtained, the system attempts to find a valid path by considering relationships in which one or more relevant concepts are involved. For each such relationship, the annotations are checked to see if the same relationship exists for all local repositories involved. Figure 3 shows that both the *calls* and *lives* relationships each exist in both DB1 and DB2 since annotations involving both databases appear on both of those links. Therefore there is only a single interpretation possible for that relationship and no further clarification is necessary. But since only one annotation, involving DB2, appears on the *rings* link, that means the relationship holds for DB2 but not for DB1. Either of the corresponding interpretations could be what the user had in mind when issuing a specific query. This is ambiguity that the system must resolve before proceeding, and it asks the user to clarify.

The gravest potential pitfall of on-the-fly dialogue is its overuse. Since it is an interruption during query processing, we wish to eliminate unnecessary interaction. We have several ways by which our system can deduce a correct interpretation. First, the profiles of other users of the same local database can be used as a guide. If most users interpret a term in the same way, the system could prevent an interruption by assuming the same interpretation for the current user. Second, the result of an on-the-fly dialogue, no matter what the user answers, may end up being moot. In our phone example, a local database may be eliminated from the search space depending on which phone type the user specified in the dialogue. If that local repository has already been eliminated due to other query constraints, then asking the user for clarification about phone types is fruitless. Third, by using the log data of past queries, it is possible to detect when the result of an on-the-fly dialogue would only result in an insignificant increase in the number of returned tuples. In this situation, the interpretation which produced the greater number of answers could be assumed with insignifi-

cant effect on processing time or user time when sorting through the results. In general, on-the-fly dialogue could be bypassed if the effect of "guessing wrong" was negligible.

4.4 Incorporating Profiles and On-the-Fly dialogue into Query Processing

We now provide an example showing how all of these elements work together to form a user-specific query interpretation. Since the focus of this paper is to show how active and passive dialogue can be integrated into current query processing techniques, we will not be discussing aspects of the query processor which do not directly affect (or are not directly affected by) the dialogue system. We will instead concentrate on those aspects of query processing in a graph-based mediator that allow profiles, on-the-fly dialogue, and annotations to resolve interpretive ambiguities. The interested reader should consult [9] for details concerning other aspects of query processing in a graph-based mediator.

For this example, we will use the mediator subset shown in Figure 6 that deals with an ecommerce network. Here a single member database, DB3, allows two possible interpretations for queries dealing with buyers. The Buyer concept can either represent those who have actually made purchases of a given product (kept in the *Product* table of DB3), or it can represent those who are just in the market to buy said product (kept in the *Want* table of DB3.) Such a distinction would be made, for example, in a name-your-own-price website such as *priceline.com.* The overall sequence of events is as follows:

1. When a new local database joins the network, the DBA uses the profile-generator toolkit to create and/or modify a questionnaire of query-related ambiguous terms, such as "Buyer".
2. When a new user wishes to use the system, s/he completes the questionnaire to create a profile.
3. The user issues distributed queries to the system in local terms while the system uses his/her profile to guide the query transformation.

When a query is issued from a member database in its own local language, a wrapper then parses the query to determine a set of mediator concepts that are relevant to the query. Call this initial set of concepts C. The processor then attempts to find a path in the mediator that includes all concepts in C. Such a path involves relationships between the concepts of C and perhaps others outside of C. Each path corresponds to a particular interpretation of the query. In the ecommerce mediator, any query dealing with *buyers* and *products* will attempt to form a connection between the two mediator nodes in some way. The edge that is used determines the interpretation. As another example, consider again the query shown in Figure 1 that asks for names and phone numbers of individuals where the *State* attribute was equal to "IL". All three mediator concepts are involved since all three are relevant to the query. The **Person** and **Phone** concepts are needed since that is what is being asked for as a result.

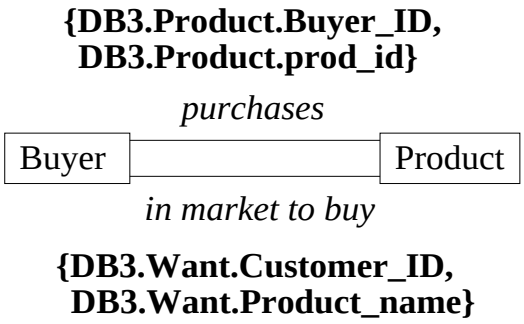

Fig. 6. Subset of Mediator used in an Ecommerce Network

The **Location** concept is relevant since it is involved in the constraint. A path connecting these three concepts that uses all three relationships (*calls*, *lives*, and *rings*) corresponds to the interpretation of looking for regular, non-mobile phones. This is because the *rings* relationship specifies the phone ringing at a particular location. A path that uses just the *calls* and *lives* relationships to connect the three concepts corresponds to the interpretation that cellular phones can be included as answers since the concept of *ringing* at a particular place is not enforced.

Dialogue is potentially introduced into query processing whenever a connection is attempted between two mediator nodes as a path is being developed. Figure 6 shows two ways that the **Buyer** and **Product** nodes could be connected: via *in market to buy* or via *purchases*. Either of these may be valid depending on the user's purpose for asking the query. For example, if the product in question was a car, then marketers of automobiles would prefer the *in market to buy* interpretation since those "buyers" are looking for a car but have not yet bought one. On the other hand, car insurance corporations would be more interested in the *purchases* relationships since it is those "buyers" who have just purchased an automobile who need insurance. But both would be asking for the same thing at their local databases: car buyers. The query processor would first consult the user's profile to see if s/he had a preference. If so, then a connection can be made immediately. By doing so, the system now knows which table in local database DB3 will be potentially supplying answers. If another local database, say DB4, had an annotation associated with *purchases* but not *in market to buy*, then a profile preferring *in market to buy* would have the added effect of eliminating DB4 from the search space since it does not conform to the user's idea of a "buyer". If a user's profile does not specify a preference (if the user chose "depends" as an answer, for example) then the user would need to be asked to specify which relationship s/he wanted via on-the-fly dialogue. In this way, a connection conforming to the user's assumptions is formed between every pair of relevant concepts, and the resulting path has a particular relevant interpretation.

DB1: Dan's Good & Services

```
Transaction(PURCHASE #, date, buyer
          product_code, quantity)

Product(PRODUCT_CODE, product_name,
       description, amt_in_stock, unit_price)
```

DB2: Al's Auctions

```
Product(PRODUCT_ID, description
       opening_bid)

Auction(AUCTION_ID, seller_id, product_id,
       high_bid, buyer_id, closing_time, num_bids)

Seller(SELLER_ID, seller_name, seller_email)

Buyer(BUYER_ID, buyer_name, buyer_email,
     current_bid, product_id)
```

DB3: Sam's Shopbot Site

```
Buyer(SAM_ID#, name, email)

Want(SAM_ID#, PRODUCT_NAME, PRICE)

Product(PROD_ID, SAM_ID#, product_name, seller,
       actual_price, qty, time_to_find)
```

Fig. 7. Schemas Used in Experimental Ecommerce Network (keys are in CAPS)

5 Results of Validation Experiments

We have completed preliminary experiments designed to validate the accuracy of our system's interpretations. The purpose of these experiments was threefold:

1. Judge the accuracy of our algorithm by seeing if query results returned by the system matched the user's idea of what "correct" results should be.
2. Determine if there was a difference in learning curve or proficiency for novice computer users versus expert users.
3. Determine if users would actually be willing to use such a system given the fact that it involves a higher degree of interaction on their part to retrieve valid answers.

Tests were administered to two groups: novice computer users and experts. "Novice" users were those who had taken a computer literacy course but were not computer experts. They had used databases before but were not expected to pursue a career in computer science. "Expert" users were undergraduate and graduate students who had majored in either computer science or computer engineering. All the experts had been exposed to databases in either coursework or industry.

The same test was given to both groups. Each user issued queries to a set of three databases that each performed on-line buying and selling in a different way. The first database kept records of goods and services sold at fixed prices,

the second recorded goods that were auctioned off to the highest bidder, and the third was a shopbot-based database that worked in a similar fashion to *priceline.com*. Their schemas are shown in Figure 7.

Users were first asked to create a profile which recorded their own opinions concerning common terms used in this domain. For example, a user was asked to define the term "Buyer" given the two interpretations presented previously in Section 4.3 and shown in Figure 6. Then, each user was shown the same set of test queries and asked to profess what they considered to be "correct" answers based on their profiles and responses to on-the-fly dialogue. The users then compared the answers returned by the system to the answers they thought would be "correct" to see if the system could properly discern their interpretation of the query. Chi-Square analysis was performed on the raw data to test for a relationship between the two user groups. The results, displayed in Table 1, show the number of users whose query interpretations matched those of the system alongside those whose interpretations did not match.

Several conclusions can be drawn from the data. The low number of matches for the first query is not surprising since the idea of multiple query interpretations is a new one to most users. This query represents a training period for the user. As time went on, the user's interpretation of the query was correctly reflected by the system's interpretation with increasing accuracy. The continually-reducing probability of independence between the two user groups indicates that expert users tended to become adept at using the system in a shorter amount of time than novices. Because non-experts are the target audience for a system of this type, we hope in the future to adjust the wording of questions in the profile generator and to provide a more robust training period using more sample queries. It is hoped that these will provide a ramp to proficiency that is neither too steep nor too long for either type of user.

Finally, interviews with test users revealed an overall willingness to put in extra effort to create a profile and answer a (limited) number of on-the-fly questions during query processing. The idea of having tailor-made queries and a system that processed them in accordance to their assumptions was attractive to 88% of novice users, even at the expense of the increased overhead.

6 Future Work

The dialogue techniques present in our mediator-based system serve as a foundation that can be adapted to most architectures, including the mediator-based systems described in Section 3. There are several directions that this research can take, from improving the existing system to work in related areas.

Several enhancements can be made to the mediator structure itself. First, our experiments revealed that not only can there exist ambiguity about local terms, but also about relationships. For example, returning to the relationships shown in Figure 6, some testers felt that *in market to buy* and *purchases* were disjoint relationships while others thought they overlapped. Those in the latter group believed that *purchases* was a subset of *in market to buy* since anyone who made

Table 1. Results from Experiment

Query	User Level	Number of Matches	Number of Mismatches	Probability of Independence
Query 1	novice	3	23	.0008
	expert	0	7	
Query 2	novice	12	14	.7001
	expert	3	4	
Query 3	novice	15	11	.3296
	expert	5	2	
Query 4	novice	9	17	.2538
	expert	4	3	
Query 5	novice	16	10	.1154
	expert	6	1	

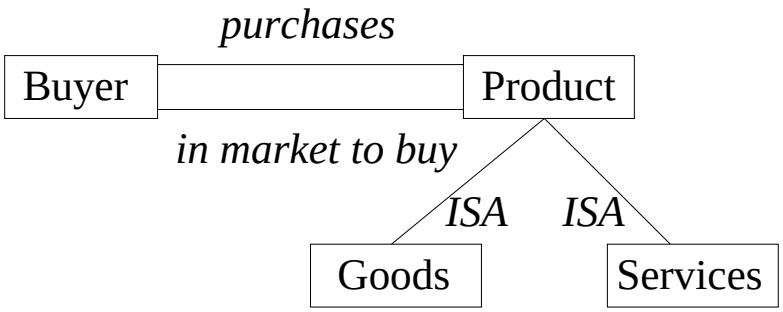

Fig. 8. Mediator with ISA relationships

a purchase must have been in the market to buy at some point too. Simply adding questions to the profiler about the relationship between relationships (which we call *meta-relationships*) is one way to explicitly model this and further control the triggering of on-the-fly dialogue.

The addition of ISA relationships to a mediator could also increase the breadth of information that could be stored. The placing of ISA relationships would not alter the necessity or functionality of our dialogue system, but its triggering could be affected. For example, consider the mediator subset shown in Figure 8. If a user were only concerned about buying actual goods and not services, and if this information was in the user's profile, then this would affect the paths that the mediator deemed valid. Given a query asking about information at the **Product** level, the processor would want to find a path, not just to the **Product** concept, but to the **Goods** concept as well, since this is the only

one valid for this query. This would imply using dialogue when a path involves a concept connected to other nodes via ISA relationships.

For complex queries with many constraints, the dialogue system could be further utilized to retrieve results that satisfy most, but not all the constraints in a query. These are called **almost answers**, and the system could return tuples that matched, say, 9 out of 10 given constraints. Of course, almost answers would be appropriately labeled to inform the user of their difference from actual answers, but for large queries, almost answers may still be of use to the user. The threshold of how close an answer would have to be to be termed an almost answer could be established in the user's profile.

The usefulness of an almost answer depends not only on how many constraints can be satisfied, but *which* constraints can be satisfied. For example, in a network about music, consider a query asking for jazz music CDs by a particular artist. An almost answer that returns music by that artist on cassette tape may be acceptable while one that returns a music CD from a different artist may not. Clearly for any query, some constraints are more important than others. Therefore the use of **filters** along with almost answers would be good improvements to implement simultaneously to our system. A filter is an addition to a profile that orders the user's preferences. In the example above, the filter would indicate that an artist has more weight in a query than the media. Being in the profile, this filter can be changed at will by the user on a query-by-query basis.

7 Conclusion

We believe that the discovery of intent in queries is an important aspect in determining if returned answers are indeed "correct" or not. If a system returns tuples based on its own static interpretation of query terms, regardless of how broadly encompassing those terms may be, the lack of consideration for the user's expectations will always leave room for misinterpretation.

We have developed and implemented a system that attempts to take the user's world assumptions into account by using profiles and on-the-fly dialogue to guide the query translation process. Using preferences specified both before and during run-time, a single query can return different sets of answers that correspond to what different users had in mind. Preliminary empirical evidence is favorable, though we hope to do more experiments to increase proficiency and decrease training time.

References

1. Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.
2. Arens, Hsu, and Knoblock. Query processing in the SIMS information mediiator. In Austin Tate, editor, *Advanced Planning Technology*, Menlo Park, CA, 1996. AAAI Press.

3. Garcia-Molina, Papakonstantinou, Quass, Rajaraman, Sagiv, Ullman, and Widom. The TSIMMIS approach to mediation: Data models and languages (extended abstract). *JIIS*, 1997.
4. Henschen, Neild, and Fernandes. An object-oriented graph traversal algorithm for data mediation. In *Proceedings of the 2nd Americas Conference on Information Systems (AIS96)*, Phoenix, AZ, August 1996.
5. Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective - slides of the invited tutorial. In *16th ACM Symposium on Principles of Database Systems*, 1997.
6. Levy, Rajaraman, and Ordille. Query-answering algorithms for information agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, December 1996.
7. Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and D. Srivastava. Answering queries using views (extended abstract). In ACM, editor, *PODS '95. Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1995, San Jose, California, May 22–25, 1995*, volume 14 of *Proceedings of the ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems*, pages 95–104, New York, NY 10036, USA, 1995. ACM Press.
8. Mena, Kashyap, Sheth, and Illarramendi. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *COOPIS*, Brussels, Belgium, June 1996.
9. Tania Neild. *The Virtual Data Integrator: An Object-Oriented Mediator for Heterogeneous Database Integration.* PhD thesis, Northwestern University, June 1999.
10. S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *The VLDB Journal, The Boxwood Press*, 1(1), July 1992.
11. Subrahmanian, Adah, Brink, Emery, Lu, Rajput, Rogers, Ross, and Ward. HERMES: A heterogeneous reasoning and mediator system. submitted for publication.
12. Tomasic, Raschid, and Valduriez. Scaling heterogeneous databases and the design of DISCO. In *Proceedings of the International Conference on Distributed Computer Systems*, 1996.
13. Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.