

Visual Querying with Ontologies for Distributed Statistical Databases

Yaxin Bi¹, David Bell¹, Joanne Lamb² and Kieran Greer³

¹School of Computer Science, Queen's University of Belfast, Belfast, BT7 1NN, UK

²CES, University of Edinburgh, St John's Land, Holyrood Road, Edinburgh, EH8 8AQ, UK

³Faculty of Informatics, University of Ulster, Newtownabbey, Co. Antrim, BT37 0QB, UK
{y.bi, da.bell}@qub.ac.uk, krc.greer@ulster.ac.uk, j.m.lamb@ed.ac.uk

Abstract. In this paper we describe a visual, ontology-based query paradigm. It has two novel features: visually specifying aggregate table queries and table layout in a single process, and providing users with an ontology guide for formulating statistical data analysis tasks as table queries which are composed of the terms defined in ontologies. We describe the role of the fundamental concept of ontology in the content representation of distributed databases with large numbers of multi-valued attributes, along with the methods and techniques developed for representing and manipulating ontologies, and for understanding semantics of database contents and query formulation and processing.

1. Introduction

The concept of “ontology” has been much used in discussions on data integration in recent years. Ontologies are useful because they provide a commonly agreed understanding of a domain for integrating separate databases through the identification of semantic connections or constraints between the data or pieces of information.

There exist some approaches for ontology-based data integration [1]. In all these cases, the ontology is mainly used to define the meaning of terms used in data sources, and to ensure the consistent use of terminology in order to cope with semantic heterogeneity reflected in heterogeneous data sources. But the way they use can be different. Three major approaches can be identified: single ontology [2], multiple ontologies [3] and hybrid approaches. From the query processing point of view, these ontologies act as query views which correspond to three query models of *global as view*, *local as view*, and *hybrid views*. Through one of these views, the user can understand the contents of diverse databases via the semantics of the data and formulate queries using terms from the ontology. As indicated in [1], one of the difficult problems related to these query models is how the ontologies can be visualized as query views in terms of their structure and content.

In the MISSON* project, we provide an ontology-based solution to the integration of distributed databases in the context of statistical databases. This work extends the MIMAD (Micro-Macro Data) model by incorporating ontologies to achieve interoperation and integration between diverse data sources instead of metadata [4, 5, 6], and by replacing a *global as view* with *local as view*. It also enhances the preliminary macro table query with a visual, table-driven query paradigm in the QBE (Query-by-Example) style

* MISSON: Multi-Agent Integration of Shared Statistical Information Over the [inter]Net

[7]. The ontologies are constructed on the basis of metadata associated with data sources and stored in a range of relations. Such ontologies are called *source ontologies*. To visualize the ontologies as views (*local as views*) and facilitate query processing, an XML-based data model is developed to hold some information from the *source ontology*. The ontology constructed in this way is called *query ontology*. The significant advantage of this model is to allow the content of data sources to be described more precisely, and in turn to be visualized to end-users on the fly and to be incorporated into a macro summary table query formulation and processing.

To achieve the best performance in the exploitation of the ontology for representing contents of databases, and query formulation and processing in an integrated environment, by taking account of the two phase query paradigm [8] and DataGuide [9], and by extending QBE, we develop a visual and ontology-based query paradigm. The underlying characteristics of our approach are as follows:

- Provide the query ontology with a well-defined XML data model, allowing the content of statistical databases with a large number of attributes to be represented and to be visualized with a hierarchical structure.
- Support aggregate table-driven querying by means of QBE
- Incorporate ontological information into query formulation and facilitate the processing queries by the agents
- Enable users to formulate complex queries with heterogeneous data sources using multi-query ontologies without prior knowledge about databases
- Adapt implicit Boolean connectives within ontology hierarchies based on two operations: *aggregation* and *generalization*.

This paper presents part of the MISSION client work and high-level query processing, the details about the system client design and ontology construction can be found in [10, 11].

2. Data model for ontologies

An ontology is defined as a shared formal conceptualization of a particular domain [12]. In practice, an ontology can be regarded as a controlled vocabulary providing a concrete specification of term names and term meanings with hierarchical structure, and can be represented in various ways, such as description logic [3] and frames [12]. To address issues in the content representation of databases, ontology visualization, query formulation and processing, we employ XML to represent ontologies, because it deals well with hierarchies, variable length attributes and large numbers of attributes. In particular, it is more suited to the generation of hierarchies on the fly, as required by the system client, than relational models.

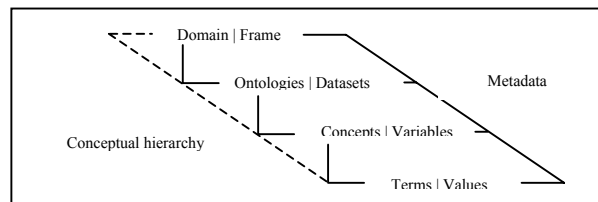


Fig. 1. Correspondence between conceptual hierarchy and metadata

In our case, an ontology is defined as a set of variables (concepts or nomenclatures) with a hierarchical structure. Each variable may consist of a set of category values. To

make an ontology more understandable, we put high level information about application domains on the ontology using the ontology constructor, called *frame*, e.g. Catewe (Comparative Analysis of Transitions from Education to Work in Europe). Fig.1 illustrates the correspondence between the general concept of ontology and the specific application. As mentioned previously, a source ontology is stored in relations and encompasses more ingredients such as mappings of nomenclatures with classification schema, etc. than a query ontology does. At the heart of this paper is the query ontology. In the following, we will interchangeably use these names, and *concept* and *variable*, etc. when we are not interested in distinguishing between them.

In the MISSION project, for representing ontologies we have refined the DTD (Document Type Definition) developed in our previous work [5]. Fig. 2 illustrates a fragment of the DTD. As we see, the FRAME is the root element, and there are four high level elements under it, i.e. TITLE, SUBJECT, DESCRIPTION, and ONTOLOGY. The first three of these represent the conceptual frame, indicating application domains to which ontologies are related. The last element, ONTOLOGY, is broken down into smaller elements: DESCRIPTION, CLASSIFICATION, and VARIABLE, to cover essential information required in the system. In particular, the element VARIABLE is used to define the meanings of concepts which are associated with the attributes within data sets, and constraints indicating the properties of the concepts.

```

<!ELEMENT FRAME (TITLE?, SUBJECT, DESCRIPTION?, ONTOLOGY+)>
<!ELEMENT ONTOLOGY (DESCRIPTION?, CLASSIFICATION?, VARIABLE+)>
<!ELEMENT VARIABLE (SET+)>
<!ATTLIST VARIABLE vartype (numeric | categorical | geographical | temporal) #REQUIRED
                    name CDATA #REQUIRED
                    mnemonic CDATA #REQUIRED>
<!ELEMENT SET EMPTY>
<!ATTLIST SET label CDATA #REQUIRED
              value CDATA #REQUIRED>

```

Fig. 2. A DTD definition for the simplified ontology (*: zero or more elements, '+': one or more elements, '?': optional, and '|': or)

```

<FRAME name = "Catewe">
  <TITLE name = "A Comparative Analysis of Transitions from Education to Work in Europe"/>
  <SUBJECT name = "Education"/>
  <ONTOLOGY name = "Catewe">
    <VARIABLE name = "Country" mnemonic="Land" vartype = "geographical">
      <SET value="Scotland"/>
      <SET value="France"/>
      <SET value="Sweden"/>
      <SET value="Netherlands"/>
      <SET value="Ireland"/>
    </VARIABLE>
    ...
  </ONTOLOGY>
</FRAME>

```

Fig. 3. An example fragment of XML data

To demonstrate how the metadata can be specified using XML, let us see the general structure of the metadata based on Fig.1. Assuming that the frame and ontology use the same name, i.e. Catewe. The ontology Catewe is composed of a set of concepts: {*Country*, *Year left school*, *Age at interview time*, *Gender*, *Type of school*}, where each concept in turn contains a set of values. For example, the *country* contains {*Ireland*, *Netherlands*, *Scotland*, *France*, *Sweden*}. Such structures perfectly match the structure that the DTD affords and can easily be specified by the DTD. Fig. 3 gives a fragment of the ontology of "Catewe" to show how the ontology is modelled (represented) in the DTD. In order to

visualize and manipulate ontology information in the user interface, we make extensive use of the DOM model (Document Object Model).

3. Posing queries against ontologies

The use of ontology to represent the content of data sources is an alternative to schema and the views-based representations. In this sense, queries will be composed using the terms defined in the ontologies, and posed against the ontologies, instead of schemas or views. In order to answer the queries, there needs to be a relationship defined between the ontology and the schema, and a methodology for ontology-based query processing that efficiently converts the query expressions derived from the ontologies into the internal expressions of the queries. Research on this aspect has not been extensively investigated yet. Some relevant issues are addressed in [13]. In our work, the solutions to the issues of relations between ontologies and database schemas, along with mappings between ontologies have been described in [14]. In this section, we discuss the other two important aspects: operations on ontologies and a table query language for expressing ontology-based queries.

3.1. Operations on ontologies

The development of effective operations on ontologies with hierarchical structure is essential for incorporating the ontologies into query formulation and processing. We develop three functions in terms of *browsing*, *aggregation* and *generalization* in our work. The first of these is mainly used for discovering fundamental concepts and relationships embedded in the hierarchy. The other two are developed for the purposes of query composition and processing. The idea behind both these functions is to map the nodes in the hierarchies of the ontologies onto the Boolean operations of intersection and union by means of the *aggregation* and *generalization* respectively, as proposed in [8]. We use these operations in different ways. In this context, an *aggregation* represents a relationship between concepts. The aggregation operations are *count*, *sum*, and *sum of square* which are performed to generate macro object tables. A *generalization* represents a combination of conceptual categories. Notice that different categories within a generalization must be disjoint. These operations constitute important operands in the function of *query construction*.

3.2. Query language

To express ontology-based queries, we adapt a table query language [14]. The table query language consists of the ten operators and the brief descriptions of the function of each operator are given in Table 1. To handle ontologies, we define two additional operators to express the aggregation and generalization. To ease query processing and enhance syntactic interoperability with ontologies encoded in XML, we have developed a markup language based on the table query language – DTD (Document Type Definition). The operators of the query language are directly used to define the tag names in the DTD. Fig.2 gives the DTD for this query language.

| Operator | Operand | Example of operand |
|----------------|--|------------------------------------|
| COMPUTE | Table | Table, graph or model |
| OF | Count | Count, sum or sum of square |
| ON | Frame | Survey, e.g. LFS |
| MERGE | Shallow | Merge t1 and t2 in shallow or deep |
| FOR | Target concept | Numerical attribute e.g. salary |
| WITH | Predicate | GENDER=Female |
| BROKEN-DOWN-BY | Cross-product of categorical attributes | GENDER by JOB |
| OVER | Geo-referenced categorical attribute(s) | Countries e.g. Scotland |
| IN | Temporal-referenced categorical attribute(s) | YEAR =1980 thru 1999 |
| Ontology | Ontology | Catewe, etc. |

Table 1. Table query language

```

<ELEMENT TQUERY (COMPUTE, OF, ON, MERGE, FOR, WITH, BROKENDOWNBY, OVER, IN,
ONTOLOGY)>
<ELEMENT COMPUTE EMPTY>
<ATTLIST COMPUTE operand CDATA #REQUIRED>
<ELEMENT ON EMPTY>
<ATTLIST ON operand CDATA #REQUIRED>
...
<ELEMENT WITH (VARIABLE+)>
<ELEMENT VARIABLE (SET+)>
<ATTLIST VARIABLE vartype (numeric | categorical | geographical | temporal) #REQUIRED>
operand (generalization) #REQUIRED
...
<ELEMENT BROKEN-DOWN-BY (VARIABLE+)>
<ATTLIST BROKEN-DOWN-BY operand (aggregation) #REQUIRED>
...

```

Fig. 4. An XML DTD for the table query language

The table query language is integrated into the system client. It provides a well-formed internal expression of ontology-based queries, and an underlying basis for converting the internal expression of the queries to executable queries and for query validation. Also this query language is visible in the query editor – a client function window – allowing users to express their information needs with minimum understanding of the query procedure and the syntax of the query language. In particular, its advantage over SQL-like languages is the capability of composing a range of the commands and structural data into objects such as frames, ontologies, attributes, and the expressiveness of statistical analysis required to generate aggregate table objects.

4. Visual query formulation

In this section, we investigate how the query ontology is involved in query formulation and processing. Before looking at a process with respect to visual specification of queries, we begin with a brief description of macro table objects.

4.1 Macro table object and table-driven queries

A macro object is defined as follows [4]:

MacroObject: $\{C_1, C_2, \dots, C_n; N_1, N_2, \dots, N_m\}$, *count*, *sum*, *sum-of-squares*
where C_1, \dots, C_n are categorical attributes with disjoint categories of values, N_1 to N_m are numeric attributes, and V_{C_1}, \dots, V_{C_n} are attribute values, i.e. $V_{C_i} = \{V_{C_{i1}}, \dots, V_{C_{ik}}\}$ is the domain

of attribute C_i . Now we stipulate that the functions of the *count*, *sum*, *sum-of-squares* are operations to be performed on the Cartesian product of the categorical values of attributes, denoted by $V_{C_1} \times \dots \times V_{C_n}$, and on numeric attributes. For example, if $C_1 = \text{Gender}$, $C_2 = \text{Employment}$ and $N_1 = \text{Salary}$, along with $V_{C_1} = \{\text{Male}, \text{Female}\}$, and $V_{C_2} = \{\text{Full-time}, \text{Part-time}\}$, then the macro table object together with the total income of individuals who are combined together is given in Table 2(a). When no numerical attributes exist, the *count* operation will be applied to C_1 and C_2 , producing the group cardinalities.

| Gender | Employment | Income SUM |
|--------|---------------|------------|
| Male | Full-time (F) | 2,122,000 |
| Male | Part-time (P) | 1,422,000 |
| Female | Full-time | 1,922,000 |
| Female | Part-time | 1,122,000 |

| | | Gender | |
|------------|---|-----------|-----------|
| | | Male | Female |
| Employment | F | 2,122,000 | 1,922,000 |
| | P | 1,422,000 | 1,122,000 |

Table 2. The macro object table (a:left, and b: right with layout it)

For queries which produce macro objects as illustrated in Table 2(a), we need a mechanism to formulate queries in a tabular QBE style. A complete configuration of table queries consists of three separate expressions in terms of *horizontal header*, *vertical stub* and *data cells*. Two of the expressions define the configuration of the x and y axes of the table, partitioning the table into rows and columns. The third expression defines the data points that hold aggregated values, corresponding to the Cartesian product of the value sets of categorical attributes. The table configuration provides the layout definition, and implicitly restricts attributes to be placed in either the *header* or *stub* only. For example, in formulating the preceding macro query, the attribute *Gender* is placed on the header and *Employment* is put on the stub, and the cells hold aggregated values produced using the *sum* operation, as illustrated in Table 2 (b).

4.2 The user interface

Fig. 5 shows a visual and interactive user interface, integrating the features of ontology visualization, visual query formulation, aggregate table-driven querying and extension of QBE to achieve three major functions: *browsing*, *query construction* and *publishing*. These functional constituents are clearly important for supporting the broader user activities in flexible ways, such as interrogating semantics of database contents. As illustrated in the screenshot, the two functions of *browsing* and *query construction* are displayed on three sub-windows. Users can interleave the functions through their views and track their progress. At the top of the query constructor, there exist two combo-boxes, one is to store numerical attributes (users can choose from them to compose queries), and the other presents a choice from the aggregation functions *count*, *sum*, *sum-of-squares*, which will be integrated into queries of macro tables.

The function of query construction provides a novel mechanism to bring query composition and table layout definition together in a single process of query formulation. Defining a layout and style on the one hand and a query on the other hand are usually seen as two different tasks which should be done in two different ways. However, the two tasks are indeed related. It is difficult to define a layout without also defining the related query. In the system client, the *query construction* component expands the QBE style as a way of query formulation, and incorporates the layout definition into the query formulation process, which acts as the client publishing function.

4.3 Visual specification

From the visual specification of queries to the executable queries, it involves three key steps: (a) converting visual specifications to the expressions of the table query language, (b) decomposing a query expressed by the table query language into a set of sub-queries expressed by SQL statements, and (c) grouping, sorting and aggregating the data drawn from different data sources and filling the results into the macro object tables. We go through these steps below.

Suppose an information need is to count “different types of school by gender within the countries of Scotland and Ireland”. The user begins with the user interface. He/she first opens the browser to obtain a list of frames, selects one frame of interest such as Catewe, and then gets a list of ontologies e.g. *Catewe Scotland*, etc. which are stored in the combo-box on the ontology browser. According to the above request, the user chooses the master ontology of Catewe. After that, as illustrated in Fig. 5(a), he/she starts to navigate the ontology information to locate the three concepts “country, type of school and gender”, respectively, and drags an individual node of *Scotland* and *Ireland* from the ontology tree and drops them on the Geographical field, drags *gender* onto the header, places *type of school* on the stub, and selects the *count* operator. When complete, the query is sent for execution. Simultaneously the visual expression formulated in the query constructor is internally converted into the expression of the table query language, a fragment of it is shown in Fig. 6.

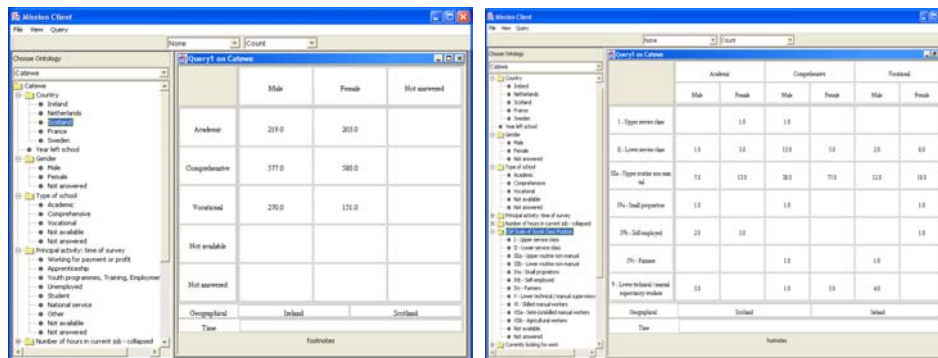


Fig. 5. The query formulation and querying result (a: left, b: right)

Fig. 6. depicts the query expression in the table language, which is derived from the visual specification of the query. The query parameters (bold pieces) dropped in the query constructor and the commands of the table query language have been integrated together, and conform to the syntax defined in Fig. 4. The conversion process from visual specifications to the expressions in the query language is determined by the commands selected, the node structures of the concepts dragged from the ontology trees and their configuration of the concepts on the query constructor. The major conversion processes include:

1. Map a selected command to the OF clause, e.g. Count.
2. Map the concepts' values constrained by the concept hierarchy from the header and stub to the WITH clause. Notice that the restrictions on the concepts and their values are implicitly linked by the *generalisation* operator, the form of query expressions is disjunction, i.e. $\langle \text{concept} = \text{value}_1 \rangle \vee \langle \text{concept} = \text{value}_2 \rangle \dots$

3. Map the concepts from the header and stub to the BREAKDOWNBY clause. In this case, the concepts on the header and stub are implicitly restricted by the *aggregation* operator, and the form of query expressions is conjunction, i.e. $\langle concept_1 \rangle \wedge \langle concept_2 \rangle \dots$.
4. The same rules are applied to the cases of nested concepts as shown in Fig. 5(b). The conjunction operations on two nested concepts are treated as the conjunction of two concepts in our cases. For example, *Gender (male and female)* is nested in *Type of School (Academic)*, so that they are a conjunction of $Gender \wedge Type\ of\ School$.

```

...
<WITH >
  <VARIABLE operand="generalization" name="Type of school" mnemonic="" vartype="geographical">
    <SET label="Academic" value=""/>
    ...
  </VARIABLE >
  <BROKENDOWNBY operand="aggregation">
    <VARIABLE name="Type of school"/>
    <VARIABLE name="Gender"/>
  </BROKENDOWNBY >
  <OVER geo="Country">
    <SET label="Scotland" value=""/>
    <SET label="Ireland" value=""/>
  </OVER >

```

Fig. 6. A query expressed in the table query language

In Figure 6, note that the attribute names, such as SCHTYPE, defined in the schemas along with their encoding values are left blank; these data will be filled during query processing. Internally the query expression is passed as an object to the agents held in the library, and then the agents perform the reformulation tasks.

5. Query reformulation

During the query process, a query composed using the ontology will be processed and reformulated into executable source-level queries. The agents first take as input a query as shown in Fig. 6, extract the frame, and the geo- and temporal-referenced categorical attributes, and match them against the frames and geo- and temporal-referenced data stored in the ontology server to prune the data sources to obtain a list of source ontologies. Then the matches of the query ontology with the list of the source ontologies are made and finally the ordinary categorical attributes are matched with the source ontologies to examine whether the attributes are included in the source ontologies to decide where the query will be executed.

If the source ontology is different from the query ontology then a heterogeneous match will be made, where categorical attributes will be reclassified and the numerical attribute will be converted by means of the mappings established beforehand. More description of heterogeneous query processing using the agents can be found in [15].

When there are enough data sources found which cover the whole query, the query will be reformulated into a set of source-level subqueries to be sent to the local databases for execution. From the definition of the macro object, we can see that the macro aggregations with the functions of *count*, *sum* and *sum-square* can be implemented using a SELECT-FROM-WHERE-GROUPBY pattern. Consequently, the task of translating the

query expressions to SQL statements is to map the commands denoted by the tags to the above pattern of the SQL clauses.

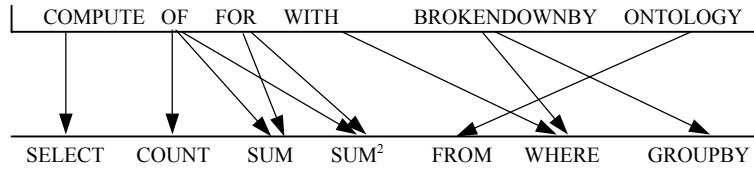


Fig. 7. Translation from the tag names to the SQL clauses

Fig.7 gives an idea of translations from the tag names defined in the table query language to the SQL clauses required by the macro aggregations. An important part is the predicate formulation in the WHERE clause. This formulation combines the query parameters and operand values (*generation* and *aggregation*) included in the VARIABLE (WITH) and BROKENDOWNBY tags to form a general form of a conjunction of disjunctions.

Fig. 8 shows the result of decomposing the query shown in Fig. 6. The query is reformulated into two subqueries. One is sent to the database that contains the “Scotland” data and the other to the “Ireland” data. In the reply process, the agent collects the answers to the query supplied by the individual database and returns to the client site. The client then presents the result to the user by filling in the macro object table as shown in Fig. 5(a).

```

Subquery1:
SELECT 0, COUNT(*), sex, schtype
FROM Scotland
WHERE ((sex=0) ∨ (sex=1) ∨ (sex=-9)) ∧ ((schtype=1) ∨ (schtype=2) ∨ (schtype=3) ∨ (schtype=-8) ∨ (schtype=-9))
GROUP BY sex, schtype
Subquery2:
...

```

Fig. 8. The subqueries derived from Fig. 6

6. Conclusion

In this work we use a visual, ontology-based query paradigm to reduce the burden on novices in understanding semantics of database contents and identifying pertinent attributes for expressing information needs. A user-friendly graphical interface built on such a query paradigm by the extension of the QBE style has fulfilled this requirement. For more advanced users, this interface offers a powerful means of composing complex queries and publishing their analysis results, thereby supporting decision making and sharing expertise.

The methods and techniques developed for this query paradigm, including the data model for representing and manipulating ontologies, ontology-based query language, query formulation and processing, and translating ontology-based queries into a set of executable sub-queries provide valuable experience for research in data integration in a broad sense. In particular, XML has been utilized effectively in this work. The approach we have taken significantly enhances the user’s ability to visualize ontologies, to express queries, and to perform validation and integrity checks. In particular, the demarcation of query parameters with the tags of the table query language provides a novel mechanism

for query expression and processing. In this sense, a query expressed in XML can be regarded as a data object. Further, the tags are treated as the operators of the query languages, in such a way that the tag names can be directly mapped to the corresponding SQL statements, significantly reducing the complexity of decomposing queries.

Acknowledgement

The work is partially supported by the MISSION project (IST 1999-10655) and partially supported by the ICONS project (IST-2001-32429), which are funded by the European Framework V. The authors would like to acknowledge the contributions made the MISSION client development team.

References

1. Wache, H., Vogele, T. Visser, U. Stuckenschmidt, H., Schuster, G., Neumann, H. and Hubner, S. (2001). Ontology-based integration of information - a survey of existing approaches. In Stuckenschmidt, H., ed., IJCAI-01 Workshop: Ontologies and Information Sharing, 108—117.
2. Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6(2/3):99--130, 1996.
3. Eduardo Mena, Arantza Illarramendi, Vipul Kashyap, Amit P. Sheth (2000). OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Pre-Existing Ontologies. *Distributed and Parallel Databases* 8(2): 223-271
4. Sadreddini, M. N. Bell, D. A. and McClean, S. I. (1992b) A Model for Integration of Raw Data and Aggregate Views in Heterogeneous Statistical databases. *Database Technology*, Vol 4, part 2, pp115-127.
5. Bi, Y., Murtagh, F. and McClean, S.I. (1999). Metadata and XML for Organising and Accessing Multiple Statistical Data Sources. *Proceedings of ASC International Conference*, Edinburgh, 393-404.
6. Scotney, B.W., McClean, S.I. & Rodgers, M. C. Optimal and Efficient Integration of Heterogeneous Summary Tables in a Distributed Database. *The Journal of Data and Knowledge Engineering*, Vol.29, pp337-350.
7. Zloof, M. (1975) Query by Example. *AFIPS*, 44.
8. Shneiderman, B. (1994). Dynamic Queries: for visual information seeking, *IEEE Software*, vol. 11, 6, 70-77.
9. Goldman, R. and Widom, J. (1997). DataGuides: Enabling Query Formulation and Optimization in Semi-structured Databases. *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, 436-445, Greece.
10. Bi, Y., Bell, D. and Lamb, J. Aggregate Table-driven Querying Via Navigation Ontologies in Distributed Statistical Databases. To appear in *Twentieth British National Conference on Databases (BNCOD)*, 2003.
11. Bi, Y., Bell, D. and Lamb, J., Greer, K. Ontology-based Access to Distributed Statistical Databases. To appear in *The Fourth International Conference on Web-Age Information Management (WAIM)*, 2003.
12. Gruber, T. (1993). A translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*. Vol. 5(2), 199-220.
13. Cui, Z., Jones, D., and O'Brien, P. (2002). Semantic B2B integration: issues in ontology-based approaches. *ACM SIGMOD Record*, Vol.31 (1), pp43-48.
14. Froeschl, K.A. (1997) *Metadata management in statistical information processing*, New York: Springer Wien.
15. McClean, S., Páircéir, R., Scotney, B. and Greer, K. (2002). A Negotiation Agent for Distributed Heterogeneous Statistical Databases. *14th International Conference on Scientific and Statistical Database Management (SSDBM)*, pp 207-216.