Quality-Extended Query Processing for Distributed Sources

Laure Berti-Equille

***Abstract*— In a classical mediator/wrapper architecture, the mediator accepts queries from users, processes them with respect to the wrappers that access to their underlying sources and return answers to the mediator for integration. To efficiently process distributed queries, the mediator must optimize the query plans. For non-collaborative data sources, both cost estimate-based optimization and quality-driven query processing are difficult to achieve because the underlying sources do not export cost information nor data quality indicators. This paper describes a new distributed query processing framework for extending the declaration of queries and for processing quality-extended queries. This processing includes the negotiation of quality contracts between the mediator and the distributed sources. We propose XQuaL, an expressive query language extension using QML[1] syntax and we present the query processor of XQuaL that achieves query decomposition with making trade-offs between cost of query and quality of result from several distributed sources.**

***Index Terms*— Source Quality, QML, Quality of Service, Negotiation, Query Decomposition.**

I. Introduction

FOR the mediator/wrapper architecture, the access to distributed information sources is carried out in a declarative way. The mediator processes the queries of the users at the global level and optimizes the query plans according to the wrappers which reach respectively their underlying data sources. In this type of environment, the sources are usually non-collaborative and do not export information describing the costs of local query processing, neither indicators of their quality of service (resource accessibility, reliability, security, etc.) nor information describing their quality of content (data accuracy, freshness, completeness, etc.).

The lack of scalability and flexibility face to the growing number and the changing structure of data sources harm seriously the effectiveness of dynamic execution of distributed queries and the quality of results. Although there are several approaches that deal with the assessment and the management of quality metadata, the dual problem of fixing the query cost and optimizing the result quality, or fixing the result quality and optimizing the query cost still remains. Querying simultaneously several information sources in a dynamic and distributed environment raises several difficult problems:

– selecting dynamically appropriate sources: different information sources may answer the global query with diverse response times, costs and result quality levels. How to define strategies for selecting adaptively the most appropriate sources or for answering the whole or parts of the global query ? Which are the criteria to select the best sources on the fly ?
– defining semantically correct distributed query plans: the result of a global query is built according to the particular order for subquery plans execution. This must combine in a coherent way both information and meta-information from the various sources ; but data quality levels are often unknown, heterogeneous (inter-source quality heterogeneity) and locally non-uniform (intra-source non-uniformity) : how to control and merge data quality indicators in a consistent way for both integrating data and quality metadata ?
– making trade-offs between the cost of query and the quality of result (including both the quality of service and the quality of content of distributed information sources): because we may accept a query result of lower quality (if it is cheaper or has a shorter response time than if the query cost is higher), it's necessary to adapt query costs to users' result quality requirements. How to measure and optimally reduce the cost and bargain query situations where the system searches for solutions that "squeeze out" more gains (in terms of result quality) than the query without quality requisites. How to develop concrete cost models to evaluate whether the expected benefits from the improved query plan and decision quality compensate for the cost of collecting and evaluating feedback from the environment during execution time ? How to adapt existing query processing techniques to environments where resource availability, allocation, quality and cost are not, by definition, decidable at compile time ?

Among the solutions proposed in the literature for some of these questions, very few approaches use the quality of service and the quality of source content to direct strategies of distributed query processing. In this context, our objectives are: (i) to extend a generic query language for describing and

[1] Quality of Service Modeling Language [4]

manipulating data and source quality contracts with different negotiation strategies, (ii) to propose a new query processing framework for selecting dynamically sources with quality-extended queries, (iii) to define appropriate database statistics, a cost model and heuristics for reducing the search space from query plan enumeration and quality-extended query negotiation, (iv) to carry out performance measures of the query execution strategies in order to validate our approach.

This paper – voluntarily limited to the two first points (i) and (ii) -- describes XQuaL (*QML-extended Query Language*) and its query processor allowing the negotiation based on the quality of sources and of data. While our general approach to query processing is typical, a number of factors associated with quality constraints specification and negotiation complicate the problem of distributed query decomposition. The challenges have been to choose an appropriate formalism for specifying quality metadata and to devise methods for query decomposition including quality-based negotiation strategies.

The paper is organized as follows. In Section II, we present previous works in the areas of quality of service (*QoS*) and of data quality. In Section III, we describe the specifications of our query language extension and its processor architecture. In Section IV, we present the definitions for the negotiation of the quality-extended queries. In Section V, we present an illustrative example. Finally, in Section VI, we conclude the paper and present our future work

II. Related Works

A. Quality of Service

Quality of service (*QoS*) has gain much attention in the field of distributed multimedia applications where it covers essentially the notion of end-to-end guarantees associated with the communication of multimedia contents over networks and their processing on computing nodes. A considerable amount of work has been done in this area to introduce models and architectures supporting such QoS guarantees in order to supersede traditional best-effort IP networks. This trend has percolated to standard architectures for open distributed processing, namely TINA [11] and more recently CORBA [18]. This narrow vision of QoS is currently generalized to extra-functional properties of distributed systems [5],[1], as QoS becomes subject to research in the middleware community [10]. QoS defines extra-functional characteristics of a system, affecting the perceived quality of the results. Such characteristics are often organized into categories. The QoS categories are generally centered on the concept of performance for distributed multimedia systems dedicated to support the audio and video transfer with various constraints of synchronization. Each category (such as reliability or security) includes several dimensions (respectively breakdown duration, number of breakdowns, etc. ; anonymity, authentication, coding, etc.). The traditional acceptance of quality of service for distributed information systems has been recently revised with a broader definition raising new problems of implementation. Traditional quality of service categories and dimensions, such as execution time or cost, are now extended to integrate the concepts of data quality, source quality as well as quality of provided services such as searching techniques, optimization strategies, data transfer or presentation. The QML language (*Quality of service Modeling Language*) has been proposed by Frølund and Koistinen [4], [5] in order to deal with the quality of service of software components in a systematic and declarative manner, allowing flexible definition of quality dimensions, quality contracts and profile constraints that we found relevant to adapt for extending SQL-like query language syntax and both querying data and quality metadata.

B. Quality of Data

Conjointly, many research works on data quality proposed definitions, conceptual models and methodologies to improve data quality in databases [30][29][26]. Most of the approaches are centered on various methods of imputation such as inferring missing data from statistical patterns of available data sets, predicting accuracy of the estimates based on the given data, data edits (automating detection and handling of outliers in data), error control [20][22][6][21]. Various methods were developed to measure the data quality provided to the users in conformance to their quality specifications or by comparison with a referential database. The use of metadata for evaluation and improvement of data quality was also recommended in [23] where producers of information were encouraged to carry out the verification, the validation, and the certification (*VV&C*) of their data. In the field of geographical information systems [9] (*ISO 15046-13, CEN prEN 287-008, FGDC)* or digital libraries *(Dublin Core, Bib-1, GILS, STARTS, Z39.50 ANSI/NISO*), most of the exchange standards propose metadata specifications for data quality which are either automatically extracted or measured by sampling from data sets. The most frequently mentioned data quality dimensions in the literature are accuracy, completeness, freshness and consistency:

- accuracy is measured by detecting the rate of incorrect values in the database,
- completeness is measured by detecting the rate of missing values in the database,
- freshness is measured by detecting the rate of obsolete values in the database,
- consistency is measured by comparison with a set of constraints by detecting the data which do not satisfy them.

But many others dimensions, metrics and measurement techniques have been proposed in the literature [6][22][28][16][13][2]. The general trend is the use of Artificial Intelligence methods (training, knowledge representation schemes, management of uncertainty, etc.) for purposes of data validation [15][14][7]. The use of machine learning techniques for data validation and correction was first presented in [19]: rules inferred from the database instances by machine learning methods were used to identify outliers in data and facilitate data validation process. Other similar approaches can be found in [24] and [25]. Utilization of statistical techniques for improving correctness of databases and introduction of a new kind of statistical integrity constraints were proposed in [12]. Statistical constraints are derived from the database instances using conventional statistical techniques (sampling, regression). Every update of the database is validated against these statistical constraints.

Probably due to the various integration issues and challenges, only very few projects focused on data quality in a distributed environment.

DWQ - Data Warehouse Quality [3], [28] is a project focused on formal modeling of information quality for optimizing the data warehouse design with quality metadata management.

HiQ B&B (*High Quality Branch and Bound Algorithm*) [17][16] is a distributed query planning algorithm that enumerates plans in such way that it usually finds the best *N* plans after computing only a fraction of the total number of plans. Upper quality bounds for partial query plans are constructed and thereby non-promising subplans are early pruned in the search tree. This integrates the source and plan quality-based selection phases to a planning algorithm that efficiently finds the top *N* plans for a query.

Other propositions concern the definition of declarative language extensions for specifying/querying quality metadata (*Q-Data* [27]) or for applying data transformations necessary to specific cleaning process (*AJAX* [8]). The prototype describes in [27] checks if the existing data are correct: the author call it data validation and cleanup by using a logical database language (*LDL++*). The system employs data validation constraints and data cleanup rules. *AJAX* [8] is an SQL extension for specifying each data transformation (such as matching, merging, mapping, clustering) for the data cleaning process. These transformations standardize data formats when possible and find pairs of records that most probably refer to the same object. The duplicate-elimination step is applied if approximate duplicate records are found and multi-table matching computes a similarity join between distinct data flows and consolidates them.

Positioned between quality of data and quality of service, our approach extends a declarative query language for taking into account, in a flexible way, aspects and constraints related to data source quality. We also adapt the query processing, particularly query decomposition in order to make trade-offs between query costs and result quality gains.

III. SPECIFICATIONS FOR QUALITY-EXTENDED QUERY LANGUAGE

Our query processing framework takes as input quality-extended queries and attempts to adapt the processing to :

1. **User preferences and quality requirements**. Adapting to user preferences includes cases where users are interested in obtaining some partial results of the query quickly with specific constraints on data quality and source quality of service. In order to meet such needs, the query processor may produce results incrementally, as they become available. The user can also classify the elements of the output in terms of importance or quality requisites and, in that case, the query evaluator has to adapt its behavior in order to produce more important or best quality results earlier. Thus, the query processor has to consider potential input from the user, such as priority ratings and quality declarations for different parts of the result.
2. **Data source statistics.** In some cases, it is not possible at compile time to gather accurate statistics about the data sources content and quality. A solution to this problem is to collect such statistical information at runtime, thereby ensuring that they are valid for the current circumstances, and to adapt query execution based on it. Techniques that adopt this policy may change the query plan when the actual statistics have become available.

In this perspective, we propose a query language extension XQuaL for defining, in a declarative way, quality-extended query and we present its query processor in the next sections of the paper.

A. *Extending Query Language with QML*

The syntax of our quality-extended query language, XQuaL is presented in Annexes and is adapted from [4] to which we refer the reader for more information. This language extension is both a Data Manipulation Language (DML) with SQL-like query parts and a Data Description Language (DDL) with contract type declaration defined into the Qwith part. Let consider the DDL extension.

A contract is a set of constraints that are defined on different user-specified dimensions. Each contract is an instance of a complex contract type. The Figure 1 proposes examples of contract types declarations (Reliability, Freshness, Completeness, and Credibility contract types) and their contract instances.

The type of the Reliability contract has four dimensions, named respectively failureMasking, serverFailure, numberOfFailures and availability. Dimension failureMasking is defined by the possible values among omission, lostResponse, noExecution, etc., of which the relation (decreasing) is interpreted such as the smallest sets of values are the preferable values for this dimension. Dimension serverFailure is defined by individual values named without order. Dimensions numberOfFailures and availability are numerical dimensions whose the first has a unit (nf/year: the number of failures per year). The Figure 1 also presents four instances of these contract types such as: S1_Reliability, S1_Freshness, S1_Completeness, S1_Credibility provided by source S1.

```
type Reliability = contract {
        failureMasking : decreasing set {omission, lostResponse,
                              noExecution, response, responseValue, stateTransition } ;  //d1
        serverFailure : enum { halt, initialState, rolledBack } ;                          //d2
        numberOfFailures : decreasing numeric nf / year ;                                  //d3
        availability : increasing numeric ;                                                //d4
} ;
type Freshness = contract {
        dataAge : decreasing numeric seconds;                                              //d5
        lastUpdate : numeric seconds ;                                                     //d6
        updateFrequency : numeric uf / month ;                                             //d7
} ;
type Completeness = contract {
        percentageOfNullValues : percentile number %;                                      //d8
        numberOfNullValuesPerQuery : numeric nnv / query ;                                 //d9
} ;
type Credibility = contract {
        reputation : enum {veryGood, good, bad, veryBad, unknown } ;                       //d10
} ;
```

```
  S1_Reliability = Reliability contract {
    failureMasking < { noExecution, response };
    serverFailure == initialState ;
    numberOfFailures < 10 nf / year ;
    availability > 0.8 ;
  } ;
  S1_Freshness = Freshness contract {
    dataAge < 4200 seconds;
    lastUpdate == 4500 seconds;
    updateFrequency == 3 uf / month;
  };
```

```
S1_Completeness = Completeness contract {
   PercentageOfNullValues == 8 %;
   NumberOfNullValuesPerQuery == 2 nnv/query;
};
S1_Credibility = Credibility contract {
   reputation == veryGood ;
};
```

Fig. 1. Example of contract types and of contract instances.

A quality-extended query (Qwith-query) is composed of extended a Qwith operator followed by different declarations (declarations) that can be declarations of types of contracts (contractTypeDeclaration), contracts (contractDeclaration) and profiles (profileDeclaration) – see Appendix.

A type of contracts (contractType) is a sequence of declarations of dimensions (dimensions), which are composed of a dimension identifier (dimName) and a dimension type (dimType). A type of dimension consists of a kind of dimension (dimSort) followed (or not) by a unit (unit).

A contract is defined (contractExpression) according to its contract type, and is identified directly by its identifier in the simple declarations or indirectly by the extended contract defined by refinement of a pre-existing contract.

The contract definition (contractDefinition) is composed of several constraints (constraints) on dimensions declared in the contract type. A constraint is composed of a constraint operator (constraintOp), which is a traditional operator of comparison, and of a target value of the constraint (dimValue). However, it is also possible to force statistically a dimension (aspect), by affecting constraints on the average value, the variance, the frequency distribution or the percentage values.

Quality can be defined as a specific combination of contracts with a relation of order on data and sources quality constraints. The Figure 2 presents our definition of quality as a complex contract type calling the previously defined contract types of Figure 1.

```
 type Quality contract {
   s : increasing set {Reliability,Freshness,Completeness,Credibility
}
      with order {Reliability<Completeness,
                  Completeness< Freshness,
                  Freshness < Credibility }
 } ;
```

Fig. 2. Example of quality contract declaration.

A profile is the set of the required contracts that are affected to one or several source(s) and/or functionalities. The profiles are useful to associate contracts to sources interfaces (or wrappers).

More precisely, a profile declaration (profileDeclaration) is defined by the identifier of the profile, the identifier of the interface and a series of expressions (profileExpression) requiring the satisfaction of contracts by entities of the targeted interfaces/wrappers (requisites).

The clause requisites binds all the interface/wrapper entities to a list of contracts, whereas the clause from... require... specifically identifies a list of interface entities to which the list of contracts definitions will be associated in the clause require.

The entities are either simple identifiers, or identifier pointers, or the identifiers qualified by the result form indicating that the list of contracts is associated with the entity result named by the identifier.

The profile declaration makes it possible to associate quality contracts with targeted source interfaces and their specific services or methods (such as query_answer_method for source S1 in Figure 3).

```
 Source_Profile for S1 = profile {
   require S1_Freshness, S1_Completeness, S1_Credibility ;
   from     S1.query_answer_method     require     S1_Reliability
contract ;
  } ;
```

Fig. 3. Example of source profile.

In the client-server architecture, a service can be called if it satisfies constraints defined in its contract profile. The service must be in conformance with the client profile, but the reverse is false. Services are in conformity with the client requirements when they are exactly the same or better.

Our approach in a standard mediator-wrapper architecture includes new functionalities such as quality source contract negotiation into the query processing. The objectives of this research are to incorporate a set of primitives extending query processing for the classical management of quality of service such as: (i) specification of quality requirements or capabilities, (ii) negotiation: reaching an agreed specification between all parties (here mediator/wrappers), (iii) admission control: comparison between the required quality and the capability to meet them, (iv) monitoring: measuring the quality actually provided by the distributed sources, (v) policing: ensuring that all parties adhere to quality contracts, (vi) maintenance: modification of system parameters to maintain the quality, and (vii) renegotiation: modification of the quality contracts and requisites.

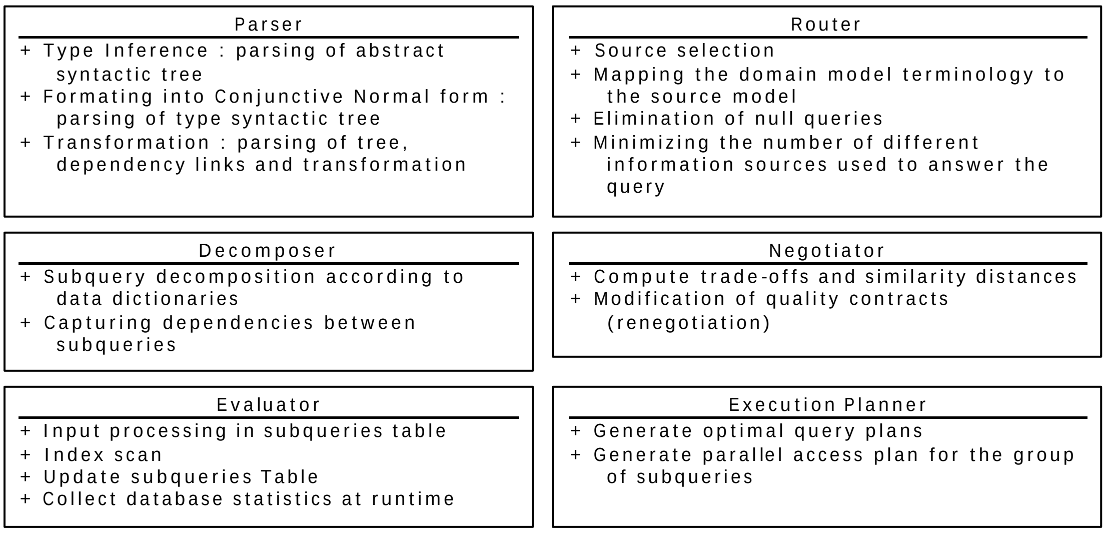
Parser
+ Type Inference : parsing of abstract syntactic tree
+ Formating into Conjunctive Normal form : parsing of type syntactic tree
+ Transformation : parsing of tree, dependency links and transformation
Router
+ Source selection
+ Mapping the domain model terminology to the source model
+ Elimination of null queries
+ Minimizing the number of different information sources used to answer the query
Decomposer
+ Subquery decomposition according to data dictionaries
+ Capturing dependencies between subqueries
Negotiator
+ Compute trade-offs and similarity distances
+ Modification of quality contracts (renegotiation)
Evaluator
+ Input processing in subqueries table
+ Index scan
+ Update subqueries Table
+ Collect database statistics at runtime
Execution Planner
+ Generate optimal query plans
+ Generate parallel access plan for the group of subqueries

Fig. 4. XQuaL Query Processor.

We have adapted these points in order to extend the query processing and have implemented specific primitives into a quality negotiator module of our query processor architecture we present in the next section dedicated to the.

B. *XQuaL Processor Architecture*

The modules of XQuaL query processor (Figure 4) are :

– ***Query parser:*** parsing the syntactic query tree
– ***Query router***: selecting relevant information sources from the available ones for answering the query
– ***Query decomposer***: decomposing the global query into a collection of subqueries, each expressed in terms of a single source model and satisfying quality requisites sent by the negotiator module
– ***Query evaluator:*** evaluating subqueries costs and collecting database statistics at runtime
– ***Query negotiator:*** making trade-offs between quality requisites and data source effective quality and renegotiating the quality contracts
– ***Query execution planner***: generating optimal plans preserving global query semantics, combining in a coherent way data and quality metadata, and take advantage of intra/inter-operators parallelism.

The user submits his query divided to two parts : the query part concerning the data and the quality extension concerning the constraints. When the *Router* receives the query, it invokes the *Contract Manager* module that returns the allowed contract types and broadcast the definition of contracts. The broadcast method invokes wrappers method to update locally metadata dictionaries of the distributed sources. Each wrapper creates a controller (monitor) in order to instantiate the contract type. The next step is to get the registered wrappers by invoking the *Contract Manager*. This method returns a set of wrappers able to answer the *Router* queries. Then, the *Router* broadcasts the QML statements to all available wrappers and wait for their response. Finally, if one of the selected wrappers satisfy the quality required it will be chosen to execute the SQL statements. If more than one wrapper respect the quality profile, the negotiator module will choose the N-best sources. Assume that the profile is too high and no wrapper can satisfy it, in this case the negotiator will renegotiate the contract terms with selected wrappers and classify them according to the negotiation strategy and will return the wrappers that conform the most to the initial contract terms. *Contract Manager, Router, Controller*, and wrappers are implemented as server classes in java.

IV. DEFINITIONS FOR NEGOTIATING QUALITY-EXTENDED QUERY

Negotiation can range over number of quantitative and qualitative dimensions of quality. Each successful negotiation requires a range of such dimensions to be resolved to the satisfaction of both parties (i.e. the querying and queried systems). To make trade-offs between query costs and quality "gains" is required in order to come to an agreement for "good quality" query results. The structure of negotiation is based almost directly on the contract used to regulate agreements which is fairly rich and covers both service and meta-service attributes. Let us now outline the formal basics of our quality-extended query negotiation model.

Let i ($i \in 1,\ldots, n$) representing the sources and j ($j \in 1,\ldots, k$) be the dimensions of quality contracts and profiles under negotiation (e.g. dataAge, failureMasking, serverFailure, ... , numberOfNullValuesPerQuery, availability etc.). Let $x_{ij} \in [min_{ij}, max_{ij}]$ be a value for quality dimension j that is acceptable to the user for querying source i. Each source has a scoring function $Score_{ij} : [min_{ij}, max_{ij}] \rightarrow [0,1]$ that gives the score of source i assigned to a value of dimension j in the range of its acceptable values. For convenience, scores are kept in the interval [0,1]. The relative importance that the user assigns to each dimension under negotiation is modeled as a weight, w_{ij}, that gives the importance of dimension j for source i. We assume the weights are normalized, i.e.

$$\sum_{1\le j\le k} w_{ij}=1, \text{ for all } i \text{ in } \{1,\ldots,n\}. \quad (1)$$

A source j scoring function for a quality contract – that is, for a value $x = (x_1,\ldots,x_k)$ in the multidimensional space defined by the dimensions value ranges – is then defined as :

$$Score_i(x)=\sum_{1\le j\le k} w_{ij}.Score_{ij}(x_j) \quad (2)$$

For analytical purposes we restrict our study to an additive and monotonically increasing or decreasing value scoring function. Negotiation consists in making trade-offs between quality profile requisites and effective quality contracts instances of sources for the query processing. The negotiator module decides to make a trade-off action when it does not wish to decrease the aspirational level (defined below and denoted q) for a given quality-extended query. Thus, the negotiator first, needs to generate some/all of the potential contracts for which it receives the score of q. And it needs to generate contracts that lie on the iso-value curve for q. Given this fact, the aim of the trade-off mechanism is to find the contract on the iso-curve that is the most preferable (since this maximizes the joint gain). More formally, the iso-curve and trade-off are defined as :

Definition 1. Given an aspirational scoring value q, the iso-curve set at level q for quality-extended query Q sent to source i, is defined as:

$$iso_i(q) = \{x \mid Score_i(x) = q\} \quad (3)$$

The heuristic we employ is to select the contract that is the most "similar" to the negotiator's last proposal (since this may be more acceptable to the user in terms of quality requirements).

Definition 2. Given a quality profile declaration, q, extending a query sent from the negotiator module to a source j and a source contract instance, *contract*, sent from the source to the negotiator with $q = Score_i(contract)$, a trade-off for the query q, sent to source i with respect to *contract* is defined as the contract that lies on the iso-value curve for the aspirational level q ; that is , a real number of the source that is the closest of the quality score required into the extended query.

$$TradeOff(contract_i,q)=\arg\left(\max\left\{Sim(z,q) \middle| z\in iso_i(\theta)\right\}\right) \quad (4)$$

where the similarity Sim, between two contracts is defined as a weighted combination of similarity of the quality dimensions.

Definition 3. The similarity between x and y over the set of quality dimensions D is defined as :

$$Sim(x, y) = \sum_{j\in D} w_j.Sim_j(x_j, y_j) \quad \text{with} \sum_{j\in D} w_j=1 \quad (5)$$

and Sim_j an appropriate similarity function for dimension j.

V. Illustrative Example

Consider the every-day life of your financial advisor who's working with on-line streaming NASDAQ quotes, offers you his experience for tracking your personal portfolio, and also has good ideas of investment based on the real-time quotes, on his knowledge of companies. Suppose he uses four main data sources : S1 - data of your bank account, S2 - the every day market report with outlook, statistics diary, and daily spotlights, S3 - connections to his usual and privileged broker's PDA and S4 - the real-time US and Intl. indices of every market and stock exchanges place. Now consider a global query sent to these four sources and assume the following scenarios:

Scenario 1: Every transaction on your bank account updates data of S1 ; S2 updates data every day ; S3 updating is very variable between one and ten times a day and S4 updates data every 15 minutes. In this case, the global query time may determine from which data source the most up-to-date data are retrieved. But up-date data does neither imply most accurate data nor most complete data.

Scenario 2: S3 is the PDA of the broker who has high credibility according to your financial advisor. S2 data may have sometimes parsing errors and may come from other web sites.

Scenario 3: S3 and S2 cover more specific sectors than S4. Information of S2 are usually less complete and less accurate than in S3 for what concerns your portfolio.

The above scenarios briefly show that the way of how and when the local data sources are queried plays an important role for the integrated result quality of global queries. They also present source dependencies and data quality dimensions such as freshness, accuracy, credibility or reliability (e.g. with PDA intermittent connections).

Actually, a user might be interested in data freshness, and another one might be rather interested in most accurate data. In both cases, it's necessary for these users to take into account, in a declarative and a flexible way, their data and source quality requirements with the global query.

Let's keep on our example with a global query involving five attributes (A1,...,A5) in the case where the sources can answer the whole or parts of the global query (see the structure of the sources in Table I) and ten quality dimensions (d1,...,d10) previously defined in Figure 1.

Several queries (Q1,...,Q4) mentioned in Table II involve the four sources. Every query is extended with the following quality statement presented in Figure 5

```
query# Qwith quality for Source_Set = profile {require
Reliability, Freshness, Completeness, Credibility};
```

Fig. 5. Example of quality-extended query.

query# is the variable representing the query number in Table 1. Reliability, Freshness, Completeness, Credibility have been previously defined as contract types in Figure 1.

The sources' quality is presented in Figure 6 over ten dimensions values in the interval [0,1]. The initial quality requisites are represented with the red line in Figure 6 that represents the aspirational scoring value.

The negotiation processing computes the trade-off between minimal query cost and maximal quality gain over the ten quality dimensions with renegotiating quality contracts and decomposes the subqueries to route to the most appropriate sources.

TABLE I.
Structure of Sources.

Sources	Attributes				
S1	a1	a2	a3	a4>30	a5
S2	a1	a2	a3		a5
S3	a1	a2<50		a4	a5
S4	a1	a2>40	a3	a4<30	

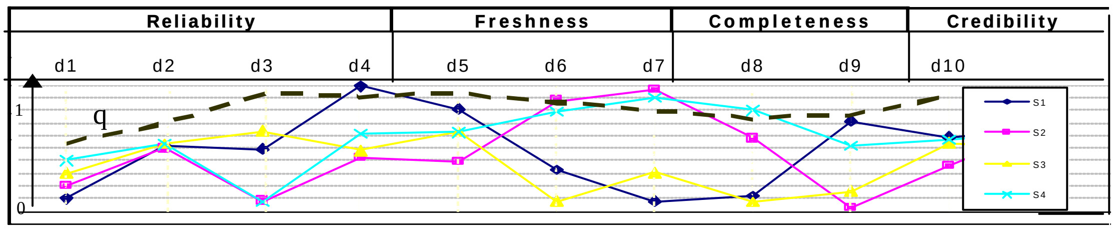

Fig. 6. Quality of Sources.

TABLE II.
Negotiation for results for quality-extended queries.

Query#	Algebraic Expression	Minimal Cost	Maximal Gain	Queried Sources
q1	Select(A1, λa1, true)	{cost(q1,s1)}	Gain$_1$	s1
q2	Select(A2, λa2, a>30)	{cost(q2,s1)}	Gain$_4$	s4
q3	Join(Q1,Q2, λa1, a2, (a1=a2))	{cost(q3,s1),cost(q3,s2),cost(q3,s3,}	Gain$_4$	s3
q4	Project(Q3, λa, a1)	{cost(q2,s4)}	Gain$_4$	s3

VI. CONCLUSION

We propose XQuaL, a query language extension including declarations of quality contracts and profiles and we briefly present the query processor that implements the algorithm of query decomposition with quality-driven negotiation. Our negotiator module is fully implemented and preliminary performance experiments are ongoing. The contribution of this work is twofold. Firstly, existent works on data quality have not proposed any declarative language for specifying and querying data as well as quality metadata: we take advantage of QML [4] in the domain of Quality of Service for flexibly extending our query language with constraints on quality of service and on quality of content of distributed data sources. Secondly, existent works on query decomposition have largely ignored the issue of source quality negotiation and trade-offs between query costs and result quality gains: our approach makes trade-offs into the query processing considering the multi-dimensional quality aspect. Our current work is now focusing on the definition of appropriate database statistics and a cost model, and we describe plan enumeration including heuristics and different strategies for quality negotiation. In most of the real world, scenarios routing need to meet stringent measures of different quality of services. It is quite natural that quality–extended query should meet a number of different and conflicting quality dimensions. Optimizing a particular objective function may sacrifice optimization of another dependent and conflicting objective. The purpose of our future work is to study the quality-based query routing problem from the perspective of multi-objective optimization.

APPENDIX : XQUAL SYNTAX

```
Qwith-query ::= query QWITH declarations ;
query ::=         ( query ) | query set-op query | query bool-
                  op query | NOT query | query constraint op
                  query | query IN query | COUNT ( query ) |
                  constant | var | sfw-query | EXISTS var
                  query : query | FORALL var query : query
set-op ::=        UNION | INTERSECT | MINUS
bool-op ::=       AND | OR
constant ::=      integer literal | real literal | quoted string
                  literal | true | false
var ::=           IDENTIFIER | $IDENTIFIER | @IDENTIFIER
sfw-query ::=     sfw-query WHERE query | SELECT projs-list
                  FROM ranges-list
projs-list ::=    projs-list, var | * | var
ranges-list ::=   ranges-list, one-range | one-range
one-range ::=     var query | path-expr
path-expression ::= path-elem.path-expression | path-elem
path-elem ::=     [from-to]query | query | path-elem[from-to]
from-to ::=       from-to:query | var
declarations ::=  declarations declaration ;
declaration ::=   contractTypeDeclaration |
                  contractDeclaration | profileDeclaration
contractTypeDeclaration ::= TYPE IDENTIFIER =
                  contractType
contractType ::=  CONTRACT { dimensions } ;
dimensions ::=    dimensions dimension | dimension
dimension ::=     dimName : dimType ;
dimName ::=       IDENTIFIER
dimType ::=       dimSort | dimSort unit
items ::=         items , IDENTIFIER | IDENTIFIER
dimSort ::=       ENUM { items } | relSem ENUM { items }
                  WITH order | SET { items } | relSem SET {
                  items } | relSem SET { items } WITH order |
                  relSem NUMERIC
relations ::=     relations , relation | relation
relation ::=      IDENTIFIER < IDENTIFIER
order ::=         ORDER { relations }
unit ::=          unit / base-unit | base-unit
base-unit ::=     % | IDENTIFIER
relSem ::=        DECREASING | INCREASING
contractDeclaration ::= IDENTIFIER = contractExpression
contractExpression ::= IDENTIFIER contractDefinition |
                  IDENTIFIER REFINED BY {constraints }
contractDefinition ::= CONTRACT { constraints }
constraints ::=   constraints constraint | constraint ;
constraint ::=    dimName constraintOp dimValue | dimName {
                  aspects }
constraintOp ::=  == | >= | <= | > | < | LIKE | !=
dimValue ::=      literal unit | literal
literal ::=       IDENTIFIER | { items } | NUMBER
aspects ::=       aspects aspect;
aspect ::=        PERCENTILE NUMBER constraintOp dimValue
                  | MEAN constraintOp dimValue | VARIANCE
                  constraintOp dimValue | FREQUENCY
                  freqRange constraintOp NUMBER %
freqRange ::=     dimValue | lRangeLimit dimValue , dimValue
                  rRangeLimit
lRangeLimit ::= [ | (
rRangeLimit ::=  ] | )
```

REFERENCES

[1] Bochmann G., and Hafid A., Some principles for quality of service management, Distributed Systems Engineering Journal, vol. 4, pp.16-27, 1997.

[2] Ballou D.P., Pazer H., Designing information systems to optimize the accuracy-timeliness tradeoff. Information Systems Research, vol. 6, no. 1, 1995.

[3] Clavanese D., DeGiacomo G., Lenzerini M., Nardi D. and Rosati R., Data integration in Datawarehousing, Tech. Rep.DWQ-UNIROMA-001, DWQ Consortium, 1997.

[4] Frølund S. and Koistinen J., QML: A Language for Quality of Service Specification. Technical Report HPL-98-10, Software Technology Laboratory, Hewlett-Packard, 1998.

[5] Frølund S. and Koistinen J., Quality of Service Aware Distributed Object Systems, In Proc. of the 5th USENIX Conf. on Object-Oriented Technologies and Systems, COOTS'99, 1999.

[6] Fox C., Levitin A., Redman T., The notion of data and its quality dimensions, Information Processing and Management, vol. 30, no. 1, 1994.

[7] Fayyad U., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. (Ed.), Advances in Knowledge Discovery and Data Mining, AAAI Press, MIT Press, 1996.

[8] Galhardas H., Florescu D., Shasha D., and Simon E., An Extensible Framework for Data Cleaning, In Proc. of the 16th Intl. Conf. on Data Engineering (ICDE 2000), 2000.

[9] Goodchild M., Jeansoulin R. (Ed.), Data quality in geographic information : from error to uncertainty, Hermès, 1998.

[10] Hanssen O., FlexiNet - QoS Investigation, Technical Report APM.1977.01.00, APM, 1997.

[11] Hamada T., Hogg S., Rajahalme J., Licciardi C., Kristiansen L., and Hansen P., Service Quality in TINA: Quality of Service Trading in Open Network Architecture, IEEE Communications Magazine, 36(8):122-130, 1998.

[12] Hou W.C., Zhang Z., Enhancing database correctness : a statistical approach. In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, 1995.

[13] Kahn B.K., Strong D.M., Wang R. Y., Information quality benchmarks: product and service performance. Com of the ACM 45(4): 184-192, 2002.

[14] Lee M. L., Lu H., Ling T.W. and Ko Y.K., Cleansing Data for Mining and Warehousing, Proc. of the 10th Intl. Conf. on Database and Expert Systems, Applications (DEXA), 1999.

[15] Maletic J. , Marcus A., Data Cleansing: Beyond Integrity Analysis, Proc. of The Conf. on Information Quality (IQ2000), Massachusetts Institute of Technology, Boston, pp. 200-209, 2000.

[16] Naumann F., Quality-Driven Query Answering for Integrated Information Systems, LNCS 2261, Springer-Verlag, 2002.

[17] Naumann F., Leser U., and Freytag J., Quality-driven integration of heterogeneous information systems. In Proc. of the 25th VLDB Conf., 1999.

[18] Object Management Group. The Common Object Request Broker: Architecture and Specification, v2.6, 2001.

[19] Parsaye K., Chignell M. Intelligent Database Tools and Applications, John Wiley & Sons, 1993.

[20] Paradice D.B., Fuerst W.L., An MIS data quality management strategy based on an optimal methodology. Journal of Information Systems, 5(1):48 - 66, 1991.

[21] Pipino P., Lee Y.W., Data quality assessment. Com. of the ACM, 45(4): 211-218, 2002.

[22] Redman T., Data quality for the information age, Artech House Publishers, 1996.

[23] Rothenberg J., Metadata to support data quality and longevity, In Proc. of the 1st IEEE Metadata Conf., 1996.

[24] Schlimmer J., ,Learning determinations and checking databases, In Proc. of the AAAI-91 Workshop on Knowledge Discovery in Databases, 1991.

[25] Schlimmer J., Self-modeling databases, IEEE Expert, 8(2):35-43, 1993.

[26] Strong D., Lee Y., Wang R., Data quality in context, Com. of the ACM, 40(5), p. 103-110, 1997.

[27] Sheth A., Wood C., Kashyap V., Q-data : Using deductive database technology to improve data quality, pp. 23-56. Kluwer Academic Press, 1993.

[28] Vassiliadis P., Data Warehouse Modeling and Quality Issues, PhD thesis, Department of Electrical and Computer Engineering, National Technical University of Athens (Greece), 2000.

[29] Wang R., A product perspective on Total Data Quality Management, Com. of the ACM, 41(2): 58-65, 1998.

[30] Wang R.Y., Storey V.C., Firth C.P., A framework for analysis of data quality research, IEEE Trans. on Knowledge and Data Engineering, 7(4):623-638, 1995.