Design and Implementation of a Peer-to-Peer Data Quality Broker

Diego Milano, Monica Scannapieco and Tiziana Catarci

Dipartimento di Informatica e Sistemistica,
Universitá degli Studi di Roma "La Sapienza",
Via Salaria 113, Rome, Italy
{milano,monscan,catarci}@dis.uniroma1.it

Abstract Data quality is becoming an increasingly important issue in environments characterized by extensive data replication. Among such environments, this paper focuses on Cooperative Information Systems (CISs), for which it is very important to declare and access quality of data. Indeed, a system in the CIS will not easily exchange data with another system without a knowledge on its quality, and cooperation becomes difficult without data exchanges. Also, when poor quality data are exchanged, there is a progressive deterioration of the quality of data stored in the whole CIS.
In this paper, we describe the detailed design and implementation of a peer-to-peer service for exchanging and improving data quality in CISs. Such a service allows to access data and related quality distributed in the CIS and improves quality of data by comparing different copies of the same data. Some experiments on real data will show the effectiveness of the service and the performance behavior.

1 Introduction

Data quality is a complex concept defined by various *dimensions* such as accuracy, currency, completeness, consistency [18]. Recent research has highlighted the importance of data quality issues in various contexts. In particular, in some specific environments characterized by extensive data replication high quality of data is a strict requirement. Among such environments, this paper focuses on Cooperative Information Systems.

1.1 Data Quality & CISs

In current government and business scenarios, organizations start cooperating in order to offer services to their customers and partners. Cooperative Information Systems (CISs) are all distributed and heterogeneous information systems that cooperate by sharing information, constraints, and goals [14]. Quality of data is a necessary requirement for a CIS. Indeed, a system in the CIS will not easily exchange data with another system without a knowledge on their quality, and cooperation becomes difficult without data exchanges. Also, when poor quality

data are exchanged, there is a progressive deterioration of the quality of data stored in the whole CIS. Moreover, when a CIS is a data integration system, data integration itself cannot be performed if data quality problems are not fixed. As an example, results of queries executed over local sources must be reconciled and merged, and quality problems resulting from a comparison of results need to be solved in order to provide the data integration system with the required information [5].

On the other hand, the high degree of data replication that characterizes a CIS can be exploited for improving data quality, as different copies of the same data may be compared in order to detect quality problems and possibly solve them.

1.2 The DaQuincis project

The DaQuinCIS architecture [1] has been designed to manage data quality in cooperative contexts, in order to avoid the spread of low-quality data and to exploit data replication for the improvement of the overall quality of cooperative data [16,11].

The DaQuinCIS architecture offers several quality-oriented services that are depicted in Figure 1.

The core component is the *Data Quality Broker*. The Data Quality Broker has two main functionalities: *(i) quality brokering*, that allows users to select data in the CIS according to their quality; *(ii) quality improvement*, that diffuses best quality copies of data in the CIS.

With reference to the quality brokering functionality, the Data Quality Broker is in essence a data integration system that allows the access to the best available quality data without having to know where such data are stored. The Data Quality Broker adopts a *wrapper-mediator* architecture, in which wrappers hide technological and model-related details of organizations, while a mediator interacts with the user, presenting a unified view of the databases on which queries can be posed.

With reference to the quality improvement functionality, when retrieving data, they can be compared and a best quality copy can be constructed. Organizations having provided low quality data are notified about higher quality data that are available in the CIS.

This paper will focus on the design and implementation features of the Data Quality Broker as a Peer-to-Peer (P2P) system. More specifically, the Data Quality Broker is implemented as a peer-to-peer distributed service: each organization hosts a copy of the Data Quality Broker that interacts with other copies and has both the functions of wrapper and mediator. While the functional specification of the Data Quality Broker is not a contribution of this paper, and has been presented in [11], its detailed design and implementation features as a P2P system

[1] The DaQuinCIS approach has been developed in the context of the project "DaQuinCIS - Methodologies and Tools for Data Quality inside Cooperative Information Systems" (`http://www.dis.uniroma1.it/~dq/`).

are a novel contribution of this paper. Moreover, we will present some results from tests made to prove the effectiveness and efficiency of our system. A first set of experiments has been performed on two real databases owned by Italian public administrations in order to show the effectiveness of the Data Quality Broker in improving their quality. Also, some experiments will be described that illustrate the performance features of the Data Quality Broker. These tests show the system's overhead due to its quality related features.

Apart from the Data Quality Broker, other fundamental components of the DaQuinCIS system are the *Quality Factory* service, responsible for evaluating the quality of data stored within organizations; the *Quality Notification Service*, which is a publish/subscribe engine allowing users to be notified on changes of the quality of specific data sets; and the *Rating Service*, that associates trust values to data sources in the CIS. These components will be not described in this paper. The interested reader may find further details in [16].

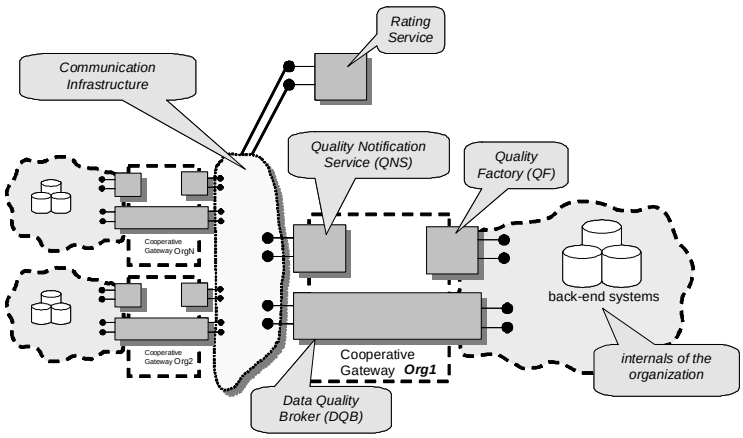


Figure 1. The DaQuinCIS architecture

1.3 Peer-to-Peer Systems

A peer-to-peer system is every distributed system in which nodes can act both as clients and servers. In other words, all nodes provide access to some of the resources they own, enabling a basic form of interoperability.

A large number of systems, having different objectives and functionalities, could be included in the basic definition given above.

In [2], an interesting classification of P2P systems can be found which distinguishes between three models, namely: *(i)* a *Decentralized Model* in which the architecture is completely decentralized, without common elements shared by peers; *(ii)* a *Centralized Model* in which peers share at least one common element (e.g. a peer search index); and *(iii)* a *Hierarchical Model*, that may be

considered as an intermediate case between the other two, in which some nodes, called super-peers, assume particular functions such as peer address index or local control.

We further classify P2P systems, on the basis of their interaction level, into *Loosely* and *Tightly* coupled systems. In the former type of systems, there are no pre-defined data exchanges. Data exchanges only consist of independent request/reply interactions between peers. In the latter type, exchanges are based on pre-defined processes that cross organizational boundaries. Such processes can even be transactional, thus originating a very tight interaction among the different organizations that participate to the system. All data exchanges are placed within a defined process.

For the Data Quality Broker, we have chosen a peer-to-peer architecture adopting a *decentralized model* with *tightly coupled* interactions. The choice of a P2P architecture for the Data Quality Broker is motivated by the need of being as less invasive as possible in introducing quality controls in a cooperative system. Indeed, cooperating organizations need to save their independency and autonomy requirements. Such requirements are well-guaranteed by the P2P paradigm which is able to support the cooperation without necessarily involving consistent re-engineering actions; in Section 6, we will better detail this point, comparing our choice with a system that instead does not adopt a P2P architecture. Decentralization and tightly coupling are both features that well-support particular type of cooperative systems, such as CISs enabling e-Government [10].

1.4 Organization of the Paper

The rest of this paper is organized as follows. In Section 2, an overview of both the component architecture and the system architecture of the Data Quality Broker is provided. In Sections 3 and 4, the modules Query Processor and Transport Engine of the Data Quality Broker are respectively described in detail. The set of performed experiments is described in Section 5. Finally, related work and conclusions are presented in Sections 6 and 7.

2 The Data Quality Broker Architecture

In this Section, we provide an overview of the Data Quality Broker from three different perspectives. First, we summarize the Data Quality Broker functionality in Section 2.1; then, in Section 2.2, the design of the Data Quality Broker as a peer-to-peer system is described, while in Section 2.3 the system architecture is detailed.

2.1 The Data Quality Broker Functionality

In the DaQuinCIS architecture, all cooperating organizations export their application data and quality data (i.e., data quality dimension values evaluated for

the application data) according to a specific data model. The model for exporting data and quality data is referred to as *Data and Data Quality (D^2Q) model* [11]. The Data Quality Broker allows users to access data in the CIS according to their quality. Specifically, the Data Quality Broker performs two tasks, namely *query processing* and *quality improvement*.

The Data Quality Broker performs *query processing* according to a global-as-view (GAV) approach by unfolding queries posed over a global schema, i.e., replacing each atom of the original query with the corresponding view on local data sources [17,9]. Both the global schema and local schemas exported by cooperating organizations are expressed according to the D^2Q model. The specific way in which the mapping is defined stems from the idea of performing a quality improvement function during the query processing step. Concept from the global schema may be defined in terms of extensionally overlapping concepts at sources. When retrieving results, data coming from different sources can be compared and a best quality copy can be constructed. Specifically, in our setting, data sources have distinct copies of the same data with different quality levels, i.e., there are *instance-level conflicts*. We resolve these conflicts at query execution time by relying on quality values associated to data: when a set of different copies of the same data are returned, we look at the associated quality values, and we select the copy to return as a result on the basis of such values. More details on the algorithm implemented for processing queries can be found in [16]. The best quality copy is also diffused to other organizations in the CIS as a *quality improvement* feedback.

2.2 The Data Quality Broker Component Architecture

The Data Quality Broker is implemented as a peer-to-peer distributed service: each organization hosts a copy of the Data Quality Broker that interacts with other copies (see Figure 2, left side). Each copy of the Data Quality Broker is internally composed by four interacting modules (see Figure 2, right side). The modules Query Processor and Transport Engine are general and can be installed without modifications in each organization. We have implemented both the Query Processor and the Transport Engine; details on their implementation will be provided in the next sections. The module Wrapper has to be customized for the specific data storage system. The module Propose Manager is implemented by each cooperating organization according to the policy chosen for taking into account quality feedbacks.

The Query Processor performs query processing, as detailed in Section 3. It unfolds queries posed over the global schema and passes local queries to the Transport Engine module. On receiving query results, a query refolding phase is performed, in order to make the execution of the global query possible. A record matching activity is then performed in order to identify all copies of same data returned as results. Then, matched results are ranked on the basis of associated quality. Finally, the Query Processor selects the best quality copy(ies) to be returned as a result.

The Wrapper translates the query from the language used by the broker to that of the specific data source. In this work the wrapper is a read-only module that accesses data and associated quality stored inside organizations without modifying them.

The Transport Engine is a communication facility that transfers queries and their results between the Query Processor module and data source wrappers.

The Propose Manager receives feedbacks sent to organizations in order to improve their data. This module can be customized by each organization according to the policy internally chosen for quality improvement. As an example, if an organization chooses to trust quality improvement feedbacks, an automatic update of databases can be performed on the basis of the better data provided by improvement notifications.

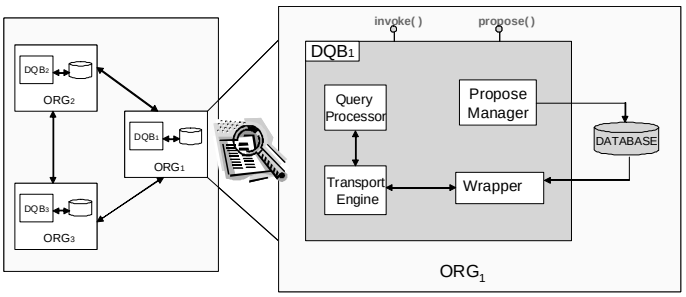


Figure 2. The Data Quality Broker as a P2P system and its internal architecture

The Query Processor is responsible for query execution. The copy of the Query Processor local to the user query, receives the query and splits it into queries local to the sources, on the basis of the defined mapping. Then, the local Query Processor also interacts with the local Transport Engine in order to send local queries to other copies of the Query Processor and receive the answers.

The Transport Engine provides general connectivity among all Data Quality Broker instances in the CIS. Copies of the Transport Engine interact with each other in two different scenarios:

- Query execution: the requesting Transport Engine sends a query to the local Transport Engine of the target data source by executing the `invoke()` operation (see Figure 2, right side) and asynchronously collects the answers.
- Quality feedback: when a requesting Transport Engine has selected the best quality result of a query, it contacts the local Transport Engines to enact quality feedback propagation. The `propose()` operation (see Figure 2, right side) is executed as a callback on each organization, with the best quality selected data as a parameter. The `propose()` can be differently implemented by each organization: a remote Transport Engine simply invokes this operation.

Another function performed by the Transport Engine is the evaluation of the *availability* of data sources that are going to be queried for data. This feature is encapsulated into the Transport Engine as it can be easily implemented exploiting Transport Engine's communication capabilities.

2.3 The Data Quality Broker System Architecture

The Data Quality Broker system architecture is based on web services technologies. To implement web services, we have chosen the J2EE 1.4 Java Platform, specifically the Java API for XML-based Remote Procedure Call (JAX-RPC) [8]. In JAX-RPC, request/response of remote methods is performed through the exchange of SOAP messages over an HTTP connection. Each copy of the Data Quality Broker is implemented by two web services, namely the Query Processing Web Service and the Data Manager Web Service as shown in Figure 3. In the Figure, the `invoke()` operation is executed by a client with a user query as input, thus making the Quality Processing Web Service call the `invoke()` operations, executed by the Data Management Web Services of the remote copies of the Data Quality Broker (see also Section 4).

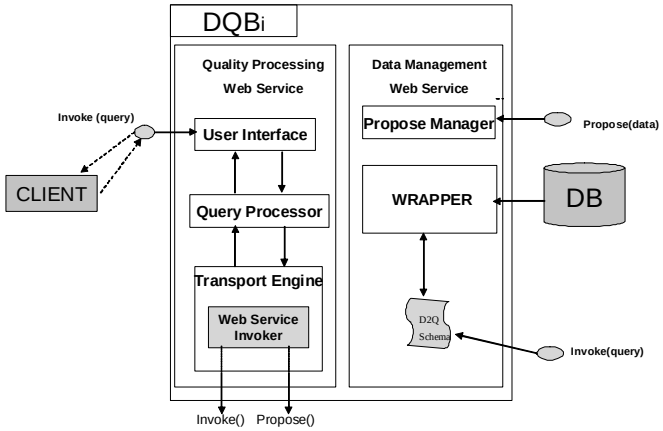


Figure 3. The Data Quality Broker system architecture

3 The Query Processor Design and Implementation

The Query Processor module of the Data Quality Broker implements the *mediation* function of a data integration architecture [19]. It performs query processing according to a GAV approach, by unfolding queries posed over a global schema. Both the global schema and local schemas exported by cooperating organizations are expressed according to the D^2Q model. The D^2Q model is a

semistructured model that enhances the semantics of the XML data model [7] in order to represent quality data. The schemas and instances of the D^2Q model are almost directly translated respectively into XML Schemas and XML documents. Such XML-based representations are then easily and intuitively queried with the XQuery language [4]. The unfolding of an XQuery query issued on the global schema can be performed on the basis of well-defined mappings with local sources. The exact definition of the mapping is beyond the scope of this paper, but the interested reader can find more details in [13].

3.1 Query Processing Steps

Query processing is performed according to the sequence of steps described in Figure 4.

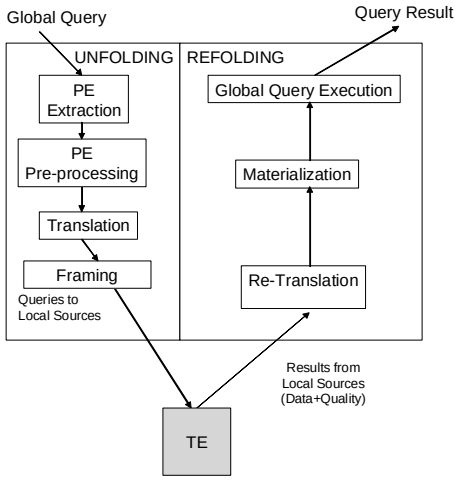


Figure 4. Sequence of steps followed in the query processing phase

The entire process may be logically divided into two phases: an *Unfolding* phase, which involves a global query and produces a set of sub-queries to be sent to local organizations, and a *Refolding* phase, which collects the results of local sub-queries execution, rewrites the global query and finally executes the global query. In the following, we will briefly revise the steps of these two phases.

The Unfolding phase starts by receiving a global query and analyzing it in order to extract those path expressions that access data from the integrated view. Only these parts of the query will be actually translated and sent to wrappers for evaluation. During the path expression extraction phase, the Query Processor looks for path expressions. The extraction is straightforward most of the times[2].

[2] In some cases, a nested expression may contain direct or indirect references to data in the global view. Such cases must be handled in a slightly different way. Our current

The result of the path expression extraction phase is a number of path expressions that have been identified and need to be translated. Before the translation phase, they are submitted to some preprocessing steps.

The preprocessing step decomposes each path expression into a set of path expressions whose concatenation yields a result equivalent to that of the original expression. The elements of this set are still expressed over the global schema alphabet, and are therefore translated into local organizations alphabets, according to the mapping specification.

After translation, sub-queries are ready to be executed at local sources. A further preliminary step is needed to make possible to re-translate their results. Usually, results of a query contain nodes and their descendants. Any information regarding their ancestors is lost. We adopt a framing mechanism in order to keep trace of ancestors and thus simplifying the retranslation phase. After retranslation, framing elements may be discarded and result fragments may be safely concatenated to form a single document.

After all the steps of the Unfolding phase have been completed, sub-queries may be passed to a Transport Engine module, which is in charge of redirecting them to local sources for execution and to subsequently collect results.

The Refolding phase starts with a step in which the received results are retranslated according to the global schema specification. Results coming from different organizations answering the same global path expression are then concatenated into a single, temporary file.

Each occurrence of a path expression previously extracted from the global query is replaced with a special path expression that accesses one of the temporary files built during the previous step. In this way, the global query is changed into a query that only uses local files, and can then be executed. Execution of a query yields a result that may contain duplicate copies of the same objects coming from different sources. For each object, a best quality representative must be chosen or constructed. For this purpose, results undergo a record matching phase that identifies semantically equivalent objects and groups them into clusters. Copies in each cluster are compared and a best quality object is either selected or constructed; more details on this process can be found in [16]. Finally, the results best fitting with the user query requirements are sent back to the user. Moreover quality feedbacks are sent to the Transport Engine that is in charge of propagating them throughout the system.

3.2 Implementation

The Query Processor has been implemented as a Java application. Figure 5 shows the main components; the phases of query processing that are executed by each component module are also represented.

---

approach is to split any path expression containing a problematic step and to treat the two parts separately. Specifically, when reverse steps are involved, they must be taken into account to perform the splitting properly.

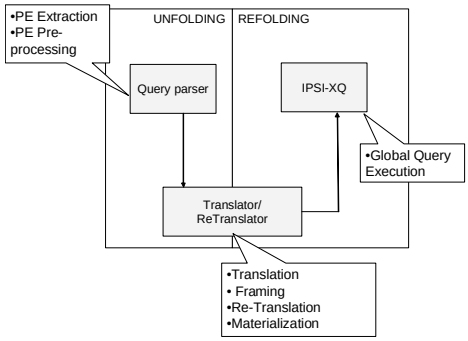


Figure 5. Implementation Modules of the Query Processor

The *Query Parser* performs the first query processing steps. To implement it, a parser for the XQuery language has been generated with the help of the JavaCC tools. The *Translation/Retranslator* module manages everything related to the translation and retranslation of queries and their results. For query execution a third-party query engine may be used. The engine used in our implementation is *IPSI-XQ* [1]. Let us note that we made IPSI-XQ quality-aware by adding some *quality functions* to it. These functions are written in XQuery, and allow to access quality data; they are simply added to the query prolog of each query submitted to the engine. Further details on this aspect may be found in [13].

4 The Transport Engine Design and Implementation

The Transport Engine component of the Data Quality Broker provides the connectivity and communication infrastructure of the DaQuinCIS system. In Figure 6 the internal components of the TE are shown; the sequence of interactions among such modules is also depicted.

The *Availability Tester* module works in background continuously executing connectivity tests with servers from other organizations. It executes a ping function on the servers in the cooperative system opening HTTP connections on them.

The *Transport Engine Interface* is the module that interfaces the Query Processor and the Transport Engine. Specifically, it uses a data structure to store queries and query results, once the latter have been gathered from each organization. The data structure is organized as an array: each element is representative of a single query execution plan and is composed by a list of queries that are specific of such a plan. Such queries are passed by the Query Processor (step 1). Then, the *Transport Engine Interface* activates the *Execute-Query* module with plans as input parameters (step 2).

The *Execute-Query* interacts with the *Availability Tester* module that performs an availability check of the sources involved in the query execution (step

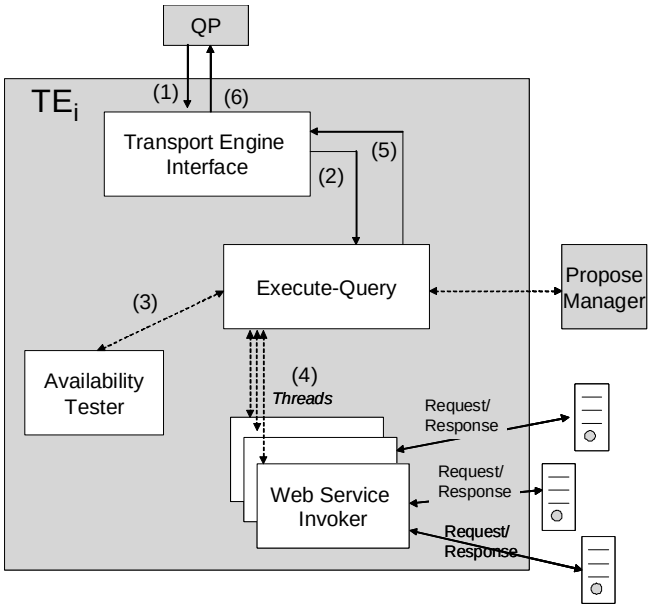


Figure 6. Internal modules of the Transport Engine of organization i

3). Then, the *Execute-Query* activates the *Web Service Invoker* module that carries out the calls to the involved organizations (step 4). The call is performed in an asynchronous way by means of suitable proxy SOAP client. Before invoking data management web services, an availability check is performed by the *Availability Tester* module. When the result of the different plans are sent back, the *Execute-Query* module stores them in a specific data structure and gives it to the *Transport Engine Interface* (step 5) that, in turn, gives it back to the Query Processor (step 6). The data structure is very similar to the input one; the main difference is the substitution of the query field with a special record containing data and associated quality provided as query answers.

Notice that the same interaction among modules shown in Figure 6 occurs when quality feedbacks need to be propagated. The Query Processor selects the best quality copies among the ones provided as query answers and then sends back the result to the *Transport Engine Interface* that activates the *Execute-Query* module with the best quality copies and the organizations to be notified about them as input parameters. The best quality copies are then sent by the *Web Service Invoker*. On the receiver organization side, the *Execute-Query* module notifies the *Propose Manager* modules of involved organizations about the better quality data available in the system, thus implementing the quality feedback functionality that the Data Quality Broker provides at query processing time.

Notice that the *Execute-Query* module, on the sender organization side, also interacts with the *Availability Tester* modules: this makes quality notification

not to be performed in a one-step process. Instead, a transaction starts that commits only when the set of sources that has to be notified is exhausted.

5 Experiments

In this Section, we first show the experimental methodology that we adopted in Section 5.1; then, in Sections 5.2 and 5.3, we show respectively quality improvement experiments and performance experiments.

5.1 Experimental Methodology

We performed two types of experiments. The first set shows the effectiveness of the Data Quality Broker to improve data quality. The second set shows some performance features of the Data Quality Broker.

We used two real data sets, each owned by an Italian public administration agency, namely:

- the first data set is owned by the Italian Social Security Agency, referred to as INPS (in Italian, Istituto Nazionale Previdenza Sociale). The size of the database is approximately 1.5 millions of records;
- the second data set is owned by the Chambers of Commerce, referred to as CoC (in Italian, Camere di Commercio). The size of the database is approximately 8 millions of records.

Some data are agency-specific information about businesses (e.g., employees social insurance taxes, tax reports, balance sheets), whereas others are common to both agencies. Common items include one or more identifiers, headquarter and branches addresses, legal form, main economic activity, number of employees and contractors, information about the owners or partners.

As far as quality improvement experiments, we have associated quality values to the INPS and CoC databases. Specifically, we have associated completeness and currency quality values to each field value. Completeness refers to the presence of a value for a mandatory field. As far as currency values, timestamps were already associated to data values in the two databases; such timestamps refer to the last date when data were reported as current. We have calculated the degree of overlapping of the two databases that is equal to about 970000 records.

As far as performance experiments, a P2P environment has been simulated. Each data source has been wrapped by a web service; such web services are deployed on different computers connected by a LAN at 100 Mbps and interacting each other using the SOAP protocol.

5.2 Quality Improvement Experiments

The experimental setting consists of the two described real data bases that are queried by a third data source that cooperates with them. We consider how this CIS behaves with regards to the quality of its data, in two specific cases. In the

first case, a "standard" system is considered; this system does not perform any quality based check or improvement action. In the second case, the CIS uses the Data Quality Broker functionality of query answering and quality improving.

Values for the frequency of queries and updates on the data bases and average query result size are derived from real use cases. We have estimated the frequency of changes in tuples stored in the two databases to be around 5000 tuples per week. Average query frequency and query result size are, respectively, of 3000 queries per week and 5000 tuples per query. In a real setting, updates are distributed over a week. Anyway, to simplify our experimental setting, we have chosen to limit updates to the beginning of each week.

We consider how the quality of the entire CIS changes throughout a period of five weeks. Note that such variations are due to both updates on the databases and exchanges of data between them. In the standard system, these exchanges are only due to queries. With the Data Quality Broker, each time a query is performed, an improvement feedback may be propagated. For both the Data Quality Broker and the standard system, we calculate the overall *Quality* of the system, as the percentage of the high quality tuples in the system. We adopt simplified quality metrics by considering that a tuple has high quality if it is complete and current on all its fields. Conversely, a tuple has low quality if it is not complete and/or current on some fields.

To clarify how the two systems reacts to updates, we have considered an update set composed by both high quality and bad quality tuples equally distributed.

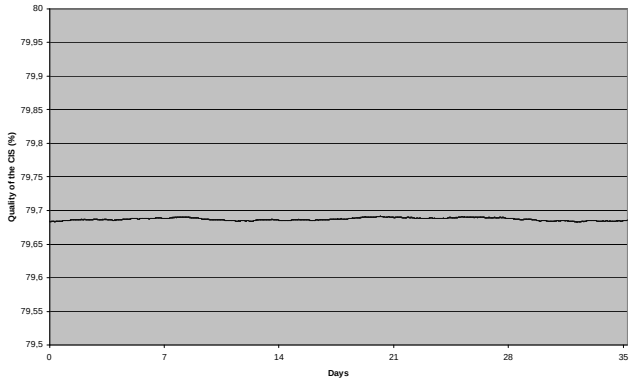
In the Figure 7, the behaviors of the Data Quality Broker and the standard system with respect to quality improvement are shown. In the standard system (Figure 7.a), the overall quality is roughly constant, due to the same number of high quality and low quality tuples spread in the system. Instead, with the Data Quality Broker (Figure 7.b), the improvement of quality in each period is enhanced by data quality feedbacks performed by the system and low quality data are prevented to spread. This causes a growing trend of the Data Quality Broker curve, in spite of low quality inserted tuples. The actual improvement is about 0.12%; given that the size of the two databases is about 9.500.000 tuples, the improvement consists of about 11.500 tuples.

As a further experiment, in Figure 8 we show the detailed behavior of the Data Quality Broker when considering a single period and varying the number of performed queries and the size of the result. This curve better shows the typical trend of the Data Quality Broker improvement due to quality feedbacks.

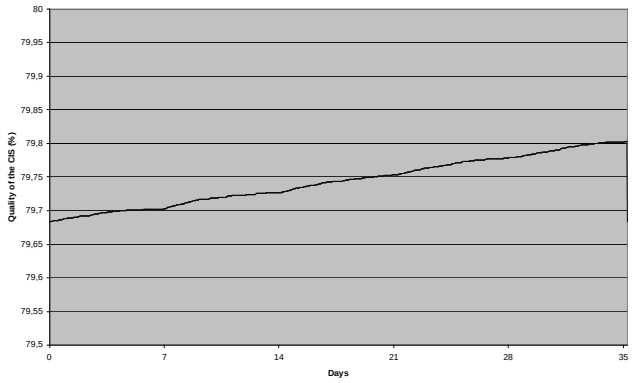
5.3 Performance Experiments

For the performance set of experiments, we have considered the Data Quality Broker and the standard system behavior with fictitious sources, in order to vary some parameters influencing performance experiments.

The first performance experiment shows the time overhead of the Data Quality Broker system with respect to the standard system. In such experiment we



(a) Data quality improvement in the standard system



(b) Data quality improvement with the Data Quality Broker

Figure 7. Data quality improvement in the standard system and with the Data Quality Broker.

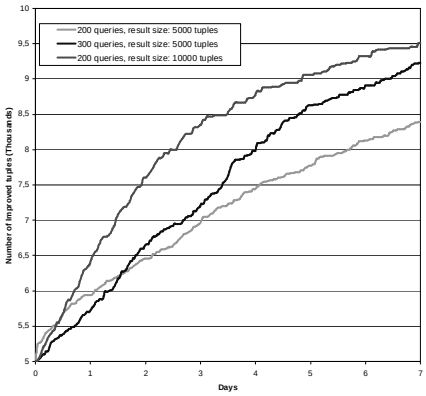


Figure 8. The Data Quality Broker improvement within a single period while varying query size and query number

draw a *normalized transaction time* defined by the fraction:

$$\frac{\texttt{DataQualityBrokerElaborationTime} - \texttt{StandardElaborationtime}}{\texttt{StandardElaborationtime}}$$

The elaboration time is the time required by the system for processing a query. The normalized transaction time is drawn when varying the degree of overlapping of data sources. The overlapping degree significantly influences the Data Quality Broker. Indeed, the Data Quality Broker accomplishes its functionalities in contexts where data sources overlap and such an overlapping can be exploited to improve the quality of data. The Figure 9 shows how the normalized transaction time varies in dependance on the percentage of data sources overlapping with two fixed query result sizes, namely q1=1000 tuples, q2= 5000 tuples. The number of overlapping sources is fixed to 3. This means that once a query is posed over the system, three sources have data that can be provided as answer to the query, though the system can have a larger number of sources. The Figure 9 shows the actual time overhead of the Data Quality Broker systems with respect to a standard system. The Data Quality Broker system has an acceptable time overhead. The worst depicted case is for the query result size q2=5000 and a percentage of overlapping equal to 40%; in such a case, there is a 50% time overhead with respect to the standard system.

The second performance experiment shows the normalized transaction time with query size varying (see Figure 10). For a fixed degree of overlapping equals to 15%, we draw the normalized transaction time for three different numbers of overlapping organizations, namely n1=3, n2=4 and n3=5. This experiment shows the behavior of the Data Quality Broker when increasing the number of organizations and the size of queries. Specifically, the normalized transaction time increases slowly with an almost linear trend. The positive result shown in

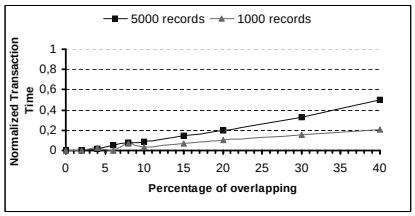


Figure 9. Normalized transaction time wrt percentage of overlapping data sources

Figure 10 is that when the number of overlapping data sources increases, the trend does not substantially change.

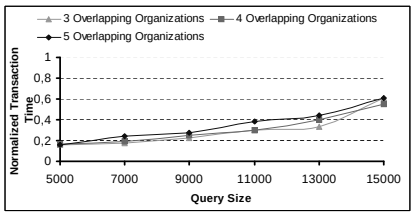


Figure 10. Normalized transaction time wrt query sizes

6 Related Work

Data quality has been explicitly addressed in a few works. In [15], an algorithm for querying for best quality data in a LAV integration system is proposed. The mapping between the local schemas and the global schema is expressed by means of assertions called Query Correspondence Assertions (QCA's). Quality values are statically associated to QCA's and to data sources. Instead, some quality values are associated to user queries at query time. In the Data Quality Broker framework, we share with [15] the idea of querying for best quality data. However, the main difference of our work with respect to [15] is the semantics of our system. Our aim is not only querying, but also improving quality of data. To such a scope, the query processing step has a specific semantics that allows to perform quality improvement on query results.

The MIT Context Interchange project (COIN) [6] is based on the idea of modeling a "context" for integrating heterogeneous sources. Such a context consists of metadata that allows for solving problems, such as instance level conflicts that may occur in the data integration phase. An integration system based on the LAV approach implements query processing with contexts. The Data Quality Broker differs mainly for considering a much more general and explicit way

of representing quality of data. Instead, the COIN approach focuses only on one aspect of data quality, namely *data interpretability*.

In [12], the basic idea is querying web data sources by selecting them on the basis of quality values on provided data. Specifically, the authors suggest to publish metadata characterizing the quality of data at the sources. Such metadata are used for ranking sources, and a language to select sources is also proposed.

In the Data Quality Broker system, we associate quality to data (at different granularity levels) rather than to a source as a whole. This makes things more difficult, but allows to pose more specific queries. For instance, the Data Quality Broker easily treats the quite common cases in which a source has some data which have a low quality and some other ones that have instead a higher quality, by making the source be an actual data provider only for better quality data. No improvement feature is present in [12].

As an e-Government initiative, the Italian Public Administration in 1999 started a project, called "Services to Businesses", which involved extensive data reconciliation and cleaning [3]. The approach followed in this project consisted of three different steps: *(i)* linking *once* the databases of three major Italian public administrations, by performing a record matching process; *(ii)* correcting matching pairs and *(iii)* maintaining such status of aligned records in the three databases by centralizing record updates and insertions only on one of the three databases. This required a substantial re-engineering of administrative processes, with high costs and many internal changes for each single administration. Differently from the approach adopted in the "Services to Businesses" project, the choice of implementing the Data Quality Broker functionality in a completely distributed way through a P2P architecture avoids bottlenecks on a single cooperating organization. Even more important, no kind of re-engineering actions need to be engaged when choosing to use the Data Quality Broker, as query answering and quality improvement can be performed with a very low impact in terms of changes on cooperating organizations.

7 Concluding Remarks

In this paper, we have described the detailed design and implementation of the Data Quality Broker service, and in particular two modules, namely the Query Processor and the Transport Engine. We have also described some experiments that validate our approach with respect to quality improvement effectiveness. Such experiments show that the Data Quality Broker succeeds in controlling and improving quality of data in a CIS. Moreover, when compared to a standard system, i.e. a system with no quality management features, the Data Quality Broker exhibits a limited performance degradation. Such a performance degradation is not a serious problem in specific scenarios, such as e-Government, in which the quality of data is the main enabling issue for service provisioning. Indeed, we remark that such scenarios are the reference ones for the DaQuinCIS system. Future works will include the deep validation based on the adoption of the proposed P2P system in some Italian e-Government pilot initiatives.

References

1. *IPSI-XQ*, Available from `http://ipsi.fhg.de/oasys/projects/ipsi-xq/index_e.html`.
2. K. Aberer and Z. Despotovic, *Managing Trust in a Peer-2-Peer Information System*, Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM'01), Atlanta, Georgia, USA, 2001.
3. M. Bertoletti, P. Missier, M. Scannapieco, P. Aimetti, and C. Batini, *Improving Government-to-Business Relationships through Data Reconciliation and Process Re-engineering*, Advances in Management Information Systems-Information Quality Monograph (AMIS-IQ) Monograph (Richard Wang, ed.), Sharpe, M.E., to appear, 2004. Shorter version also in ICIQ 2002.
4. S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, and J. Simèon, *XQuery 1.0: An XML Query Language*, W3C Working Draft. Available from `http:///www.w3.org/TR/xquery`, November 2003.
5. M. Bouzeghoub and M. Lenzerini (editors), *Special Issue on Data Extraction, Cleaning, and Reconciliation*, Information Systems **26** (2001), no. 8.
6. S. Bressan, C.H. Goh, K. Fynn, M.J. Jakobisiak, K. Hussein, K.B. Kon, T. Lee, S.E. Madnick, T. Pena, J. Qu, A.W. Shum, and M. Siegel, *The COntext INterchange Mediator Prototype*, Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1997), Tucson, Arizona, USA, 1997.
7. M.F. Fernandez, A. Malhotra, J. Marsh, M. Nagy, and N. Walshand, *XQuery 1.0 and XPath 2.0 Data Model*, W3C Working Draft. Available from `http:///www.w3.org/TR/query-datamodel`, November 2002.
8. JSR-101 Expert Group, *Java(tm) api for xml-based remote procedure call (jax-rpc) specification version 1.1*, Sun Microsystems, Inc., October 2003.
9. M. Lenzerini, *Data Integration: A Theoretical Perspective*, Proceedings of the 21st ACM Symposium on Principles of Database Systems (PODS 2002), Madison, Wisconsin, USA, 2002.
10. M. Mecella and C. Batini, *A Review of the First Cooperative Projects in the Italian* e-*Government Initiative*, Proceedings of the 1st IFIP Conference on *e*-Business, *e*-Commerce, *e*-Government, Zurich, Switzerland, 2001.
11. M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, and C. Batini, *The DaQuinCIS Broker: Querying Data and their Quality in Cooperative Information Systems*, Journal of Data Semantics **1** (2003), no. 1. Shorter version also appeared in CoopIS 2002.
12. G. Mihaila, L. Raschid, and M. Vidal, *Using Quality of Data Metadata for Source Selection and Ranking*, Proceedings of the 3rd International Workshop on the Web and Databases (WebDB'00), Dallas, Texas, 2000.
13. D. Milano, M. Scannapieco, and T. Catarci, *Quality-driven Query Processing of XQuery Queries*, Submitted to International Conference, 2004.
14. J. Mylopoulos and M.P. Papazoglou (eds.), *Cooperative Information Systems (Special Issue)*, IEEE Expert Intelligent Systems & Their Applications **12** (1997), no. 5.
15. F. Naumann, U. Leser, and J.C. Freytag, *Quality-driven Integration of Heterogenous Information Systems*, Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, UK, 1999.
16. M. Scannapieco, A. Virgillito, M. Marchetti, M. Mecella, and R. Baldoni, *The DaQuinCIS architecture: a Platform for Exchanging and Improving Data Quality in Cooperative Information Systems*, Information Systems (to appear, 2004).

17. J.D. Ullman, *Information Integration Using Logical Views*, Proceedings of the 6th International Conference on Database Theory (ICDT '97), Delphi, Greece, 1997.
18. R.Y. Wang and D.M. Strong, *Beyond Accuracy: What Data Quality Means to Data Consumers*, Journal of Management Information Systems **12** (1996), no. 4.
19. G. Wiederhold, *Mediators in the Architecture of Future Information Systems*, IEEE Computer **25** (1992), no. 3.