

---

# DAQUNCIS

Metodologie e Strumenti per la Qualità dei Dati in Sistemi Informativi Cooperativi

<http://www.dis.uniroma1.it/~dq/>

Programma di Ricerca Cofinanziato dal MIUR (Esercizio 2001)

---

## DL6.B *DaQuinCIS* Architecture: Data Quality Broker experimental validation.

LUCA DE SANTIS, DIEGO MILANO, GABRIELE PALMIERI, MONICA SCANNAPIECO

### Sommario

In this document, we describe an experimental phase that some of the modules of the *DaQuinCIS* platform, specifically the Data Quality Broker, have been submitted to. This test phase validates the feasibility and effectiveness of the *DaQuinCIS* approach by applying it to real databases; it also quantitatively evaluates the efficiency of the system. In the proposed environment, the platform complements a periodical record matching activity with an "on-line" quality improvement, performed at query processing time.

<b>Data</b>	
<b>Tipo di prodotto</b>	Rapporto tecnico
<b>Numero di pagine</b>	13
<b>Unità responsabile</b>	RM
<b>Unità coinvolte</b>	RM
<b>Autore da contattare</b>	

# 1 Introduction

In this document, we describe an experimental environment and a series of tests that have been set up to validate the approach to data quality improvement proposed in the *DaQuinCIS* project. More specifically, we show a combined approach, based on both a periodical record matching activity and an “on-line” quality improvement activity performed at query processing time, measure quality improvement obtained through this method and estimate the network overload coming from the *DaQuinCIS* architecture, compared against the workload of a standard cooperative system. All tests are made using real databases.

## 1.1 Outline

The document is organized as follows. In section Section2 we revise the *DaQuinCIS* approach to data quality improvement. Section 3 provides an overview of the architecture of *DaQuinCIS*. Section 4 describes the record matching algorithm. Section 5 describes the new approach to quality improvement at query-time used in *DaQuinCIS*. Section 6 discusses the implemented system. Section 7 describes our experimental results.

## 2 The *DaQuinCIS* data quality improvement strategy

In CIS’s different copies of the same data are typically stored by multiple sources and can be compared in order to detect quality problems and possibly solve them. The basic idea of the *DaQuinCIS* approach is to improve quality of data replicated in a CIS through extensive data comparisons: whereas copies of same data are different because of data errors, comparisons help to reconcile such copies.

The *DaQuinCIS* approach proposes to improve data quality in two distinct steps:

- *Periodical Record Matching*: a distributed record matching algorithm is run on data sets stored by the organizations in the CIS. The analyzed data sets are the ones that cooperating organizations exchange each other within cooperative processes to which they participate [1].
- *Query-time improvement*: data comparisons are performed at query processing time by comparing data received as answers to queries.

This approach is supported by a platform implemented as a set of peer to peer services for quality improvement and maintenance in CISs.

In the *DaQuinCIS* approach, we consider a real scenario as a reference setting, namely the Italian Public Administration (PA).

As an e-Government initiative, the Italian PA 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 PA, 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 *DaQuinCIS* approach does not create any bottleneck on a single cooperating organization, since the improvement is managed in a completely distributed way. Moreover, no kind of re-engineering actions need to be engaged, as the quality improvement actions are enacted according to a non-invasive policy.

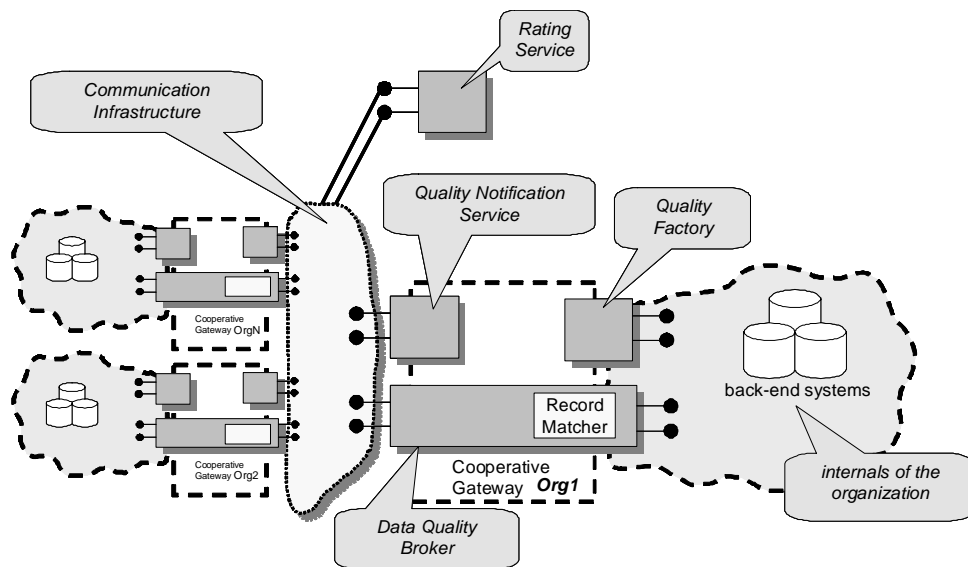


Figure 1: The *DaQuinCIS* architecture

### 3 The *DaQuinCIS* architecture

The *DaQuinCIS* architecture allows to exchange data and associated quality and exploits data replication to improve the overall quality of cooperative data.

Each organization offers services to other organizations and also specific services to its internal back-end systems. Such services are deployed on *cooperative gateways* that interface each cooperating organization with the other ones. Moreover, the communication infrastructure itself offers some specific services. Services deployed on cooperative gateways are all identical and peer, i.e., they are instances of the same software artifacts, and act both as servers and clients of the other peers depending on the specific activities to be carried out. The overall architecture is depicted in Figure 1.

Organizations export data and quality data according to a common model, referred to as *Data and Data Quality ( $D^2Q$ ) model*. It includes the definitions of (i) constructs to represent data, (ii) a common set of data quality properties, (iii) constructs to represent them and (iv) the association between data and quality data. More details on the  $D^2Q$  model can be found in [15].

In order to produce data and quality data according to the  $D^2Q$  model, each organization deploys on its cooperative gateway a **Quality Factory** service that is responsible for evaluating the quality of its own data. More details on the Quality Factory can be found in [4].

The **Data Quality Broker** poses, on behalf of a requesting user, a data request over other cooperating organizations, also specifying a set of quality requirements that the desired data have to satisfy; this is referred to as *quality brokering functionality*. Details on the brokering functionality can be found in [13].

In this document, we focus on a different functionality of the *DaQuinCIS* namely the *quality improvement functionality* that is carried on in two phases, namely: (i) record matching is periodically performed by the **Record Matcher** component; (ii) comparisons among same data are performed at query time. Different copies of the same data received as responses to a query are reconciled and a best-quality value is selected and proposed to organizations.

The record matching algorithm implemented by the Record Matcher was first proposed in [2] and is summarized in Section 4.

With respect to the functionality of quality improvement at query time, the Data Quality Broker is in essence a peer-to-peer data integration system with a semantics that allows to perform quality improvement. The newly introduced quality improvement functionality of the

Data Quality Broker is described in Section 5.

The **Quality Notification Service** is a publish/subscribe engine used as a quality message bus between services and/or organizations. More specifically, it allows quality-based subscriptions for users to be notified on changes of the quality of data. For example, an organization may want to be notified if the quality of some data it uses degrades below a certain threshold, or when high quality data are available. Also the Quality Notification Service is deployed as a peer-to-peer system. More details on the Quality Notification Service can be found in [12].

The **Rating Service** associates trust values to each data source in the CIS. These values are used to determine how much an organization can be trusted with respect to provided data. The interested reader can refer to [5] for more details.

## 4 Record Matching

Record Matching, also known as Record Linkage [6] or Object Identity problem [18], is the problem of identifying if two records are related to the same real world entity. In the *DaQuinCIS* platform, a record matching activity is performed in two phases:

- a periodical record matching is run in order to align different copies of the same entities that are present in data sources. The algorithm runs on data stored on distributed sources, according to a policy based on [7].
- Record matching also supports the query processing phase by identifying same instances in query results returned by each data source (this step will be detailed in the next section).

The Record Matcher module is a component of the Data Quality Broker as shown in Figure 1. The Record Matcher implements a method for record matching based on the quality data exported by cooperating organizations. The proposed method is an improvement of the Sorted Neighborhood Method (SNM) [8]. The SNM consists of three distinct steps:

- Choice of the matching key. A key needs to be chosen in order to sort records that are going to be matched. The choice of the key is a fundamental step of the matching process, as the results of the matching depend on how much potentially matching records are close to each other after the sorting step.
- Sorting of records according to the chosen key.
- Moving of a fixed size window through the list of records and comparisons only of the records included in the window. Each couple of records in the window is compared in order to decide if the two records match or not.

The algorithm used in *DaQuinCIS* is different from SNM with reference to the following aspects:

- Automation of the matching key choice phase. This phase is typically realized by a human expert, known as *key designer*. Instead, we suggest using quality data exported by organizations in order to select such a key. The details of the proposed key selection algorithm are described in Section 4.1.
- The decision about the matching of two records is taken in a domain independent way by considering a function that normalizes a classic edit distance function upon string lengths. Such a function and the procedure used to declare that two records are duplicate are described in Section 4.2.

## 4.1 Automatic Choice of the Matching Key

We propose to exploit quality data exported by each cooperating organization in order to automatically choose the matching key. The idea is to choose a high "quality" key. Let us consider as an example the choice of a key with a low completeness value; after a sorting on the basis of such a key, the potential matching records can be not close to each other, due to null values. Similar considerations can be made also in the case of low accuracy or low consistency of the chosen key; a low accurate or low consistent key does not allow to have potential matching records close to each other. Therefore, we evaluate the quality of the matching key in terms of accuracy, consistency and completeness.

Besides quality of data, the other element influencing the choice of the key is the *identification power*. Let us consider as an example a record **Citizen** with fields **Surname**, **Name**, **Address** and **Sex**. Though the field **Sex** may have the greater quality value, it would not be appropriate as matching key because it can have only two values, i.e. **Male** and **Female**, and thus all records are simply divided into two sets, without having similar records close to each other.

Basing on such considerations, we introduce a parameter called **Identification power of the attribute j** ( $ip_j$ ), in order to evaluate the discriminating power of record attributes.

**Definition 4.1 [ $eq_j$  relation]** Given two records  $r_1$  and  $r_2$ , and given an attribute  $j$  of the two records, we define the equivalence relation  $eq_j$  such that  $r_1 eq_j r_2$  iff  $r_1.j=r_2.j$ , i.e. the value of the attribute  $j$  of the record  $r_1$  is equal to the value of the attribute  $j$  of the record  $r_2$ .

**Definition 4.2 [ $ip_j$  identification power]** The *Identification Power of the attribute j*  $ip_j$  is defined as the number of distinct equivalence classes originated by the relation  $eq_j$  applied on the total of records  $\div$  Total number of records

Beside considering the identification power of a single attribute, it is also possible to consider the identification power of sets of attributes, for which the given definition can be easily extended. Notice that it is trivial to show that the identification power of the attributes  $(x, y)$  is equal to the identification power of the attributes  $(y, x)$ .

The choice of the matching key is performed on the basis of a function that combines quality of attributes with their identification power.

If  $dq_j$  is the overall quality value and  $ip_j$  is the identification power, we introduce the function  $k_j$ , such that:

$$k_j = dq_j * ip_j$$

Let us consider all the attributes  $j$  of records, the steps to calculate the matching key are the following ones:

- Computation of the Data Quality of the attribute  $j$ .
- Computation of the Identification Power of the attribute  $j$ .
- Computation of the function  $k_j$ .
- Selection of the matching key as  $max\{k_j\}$ .

The selection of a set of attributes to construct the key is also possible and the computation of the Data Quality and the Identification Power can be easily extended to such cases.

## 4.2 Matching Decision

The method we propose for matching decision is based on a specific edit distance function; string or edit distance functions consider the amount of difference between strings of symbols. We have chosen the Levenshtein distance [11], which is a well known early edit distance where the

difference between two text strings is simply the number of insertions, deletions, or substitutions of letters to transform one string into another.

The function we use for deciding if two strings  $S1$  and  $S2$  are the same is also dependent from the lengths of the two strings as follows:

$$f(S1, S2) = \frac{\max(\text{length}(S1), \text{length}(S2)) - LD(S1, S2)}{\max(\text{length}(S1), \text{length}(S2))}$$

According to such a function, we normalize the value of the Levenshtein distance on the maximum between the lengths of the two strings, i.e. the function  $f$  is 0 if the strings are completely different, 1 if the strings are completely equal.

The procedure we propose to decide if two records are duplicate is the following:

- the function  $f$  is applied to the values of a same attribute in the two records. If the result is greater than a fixed threshold  $T1$ , the two values are considered equal; we call  $T1$  *field similarity threshold*.
- If the number of equal pairs of values in the two records is greater than a threshold  $T2$ , then the two records are considered as duplicates; we call  $T2$  *record similarity threshold*.

The thresholds  $T1$  and  $T2$  have to be fixed experimentally. Specifically, if choosing an exact matching policy,  $T1=1$  and  $T2=\text{Total number of attributes composing records}$ . Otherwise, if choosing an approximate matching policy, the two thresholds can be adjusted according to the required strictness of the matching policy.

We considered recall and precision metrics for the proposed record matching algorithm on experimental data sets. The results are described in [5] and show the effectiveness of the method with respect to the basic Sorted Neighborhood Method.

The final result of the record matching algorithm is a partition of records into a set of *clusters*, each one consisting of records referring to the same real world object.

## 5 Query-Time Quality Improvement

In this section we describe the query processing phase as performed by the Data Quality Broker (DQB) component of the *DaQuin CIS* platform. Specifically, we focus on details concerning quality improvement within query processing.

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, 10]. 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. This means that each concept from the global schema is defined in terms of extensionally overlapping concepts at sources. Therefore, when retrieving data, they 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 construct the copy to return as a result on the basis of such values. The best quality copy is also diffused to other organizations in the CIS as a quality improvement feedback.

The detailed steps to answer a query and construct the best quality result are the following (see Figure 2):

1. let  $Q$  be a query posed on the the global schema  $\mathcal{G}$ ;

2. The query  $\mathcal{Q}$  is unfolded according to the static mapping that defines each concept of the global schema in terms of the local sources; such a mapping is defined in order to retrieve all copies of same data that are available in the CIS. Therefore, the query  $\mathcal{Q}$  is decomposed in  $\mathcal{Q}_1, \dots, \mathcal{Q}_k$  queries to be posed over local sources;
3. The execution of the queries  $\mathcal{Q}_1, \dots, \mathcal{Q}_k$  returns a set of results  $\mathcal{R}_1, \dots, \mathcal{R}_k$ .

On such a set an *extensional correspondence* property is checked. Specifically, the record matching algorithm described in the previous section is run on the set  $\mathcal{R}_1 \cup \mathcal{R}_1 \cup, \dots, \cup \mathcal{R}_k$ .

The result of this step is the construction of set of clusters composed by tuples referring to same real world objects, namely  $\mathcal{C}^1, \dots, \mathcal{C}^z$ .

4. The result to be returned is built by relying on a best quality default semantics. For each cluster, a best quality representative is either selected or constructed.

More specifically:

- Let us consider the generic cluster  $\mathcal{C}^i = \{\tau_j^i\}$ , with  $1 \leq j \leq \|\mathcal{C}^i\|$ , being  $\tau$  a tuple in the results  $\mathcal{R}_1, \dots, \mathcal{R}_k$  and  $\|\mathcal{C}^i\|$  the cardinality of  $\mathcal{C}^i$ .
- Let  $\tau_j^i = \{ \langle f_{j,m}, q_{j,m} \rangle \}$  be a generic tuple of the cluster in which a quality value  $q$  is associated to each field value  $f$ ;  $m$  can range from 1 to  $w$ , being  $w$  the number of the fields of the tuple.
- The following cases can occur:
  - if  $\forall m$  and  $\forall h \in \{1, \|\mathcal{C}^i\|\}$ ,  
 $q_{j,m} > q_{h,m}$   
 then  $\tau_j^i$  is selected as the best quality representative of the cluster;
  - otherwise, the best quality representative is constructed by selecting field values  $f_{j,m}$  such that, for a fixed  $m$  and  $\forall h \in \{1, \|\mathcal{C}^i\|\}$ ,  
 $q_{j,m} = \max\{q_{h,m}\}$ .

Let us note that each quality value  $q$  is a vector of quality values corresponding to the different quality dimensions. For instance,  $q$  can include values for accuracy, completeness, consistency and currency. These dimensions have potentially different scales, therefore a scaling problem occurs. Once scaled such vectors need to be ranked, therefore also a ranking method must be applied. Both scaling and ranking problems have well-known solutions (e.g., multi-attribute decision making methods, like AHP [14], see also later in this section).

5. Once representatives for each cluster have been selected, the result  $\mathcal{R}$  is constructed as the union of all cluster representatives.  $\mathcal{R}$  is returned as a result to the query  $\mathcal{Q}$  and it is also proposed as a best quality set to organizations, having provided lower quality results.

Note that, in the above, we have simplified the semantics from schema heterogeneities issues that are out of the scope of the present document; nevertheless such issues have been considered in the *DaQuinCIS* system [16].

## 6 The Data Quality Broker: Design and Implementation Issues

In this section we summarize the main design and implementation choices for the Data Quality Broker.

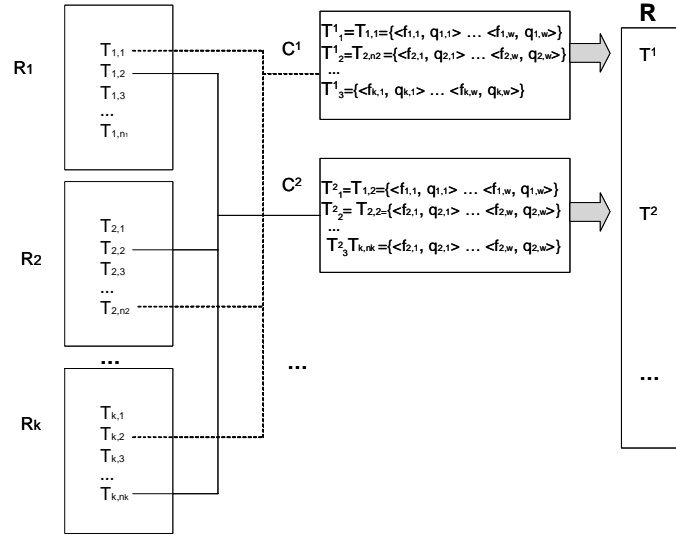


Figure 2: Best quality copy construction at query time.

### 6.1 Internal Modules of the Data Quality Broker

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. It is internally composed of five interacting modules (Figure 3). The modules Query Engine and Transport Engine are general and can be installed without modifications in each organization. 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 module Comparator, used by the Query Engine in order to compare different quality values, can also be customized to implement different criteria for quality comparison.

**Query Engine** : performs query processing. 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 possible the execution of the global query. Then, a record matching activity is performed and matched results are passed to the Comparator module that ranks results on the basis of associated quality. Finally, the Query Engine filters best quality result(s) to be sent back to users.

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

**Transport Engine** : communication facility that transfers queries and their results between the Query Engine module and data source wrappers.

**Propose Manager** : it receives improvement 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 the 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.

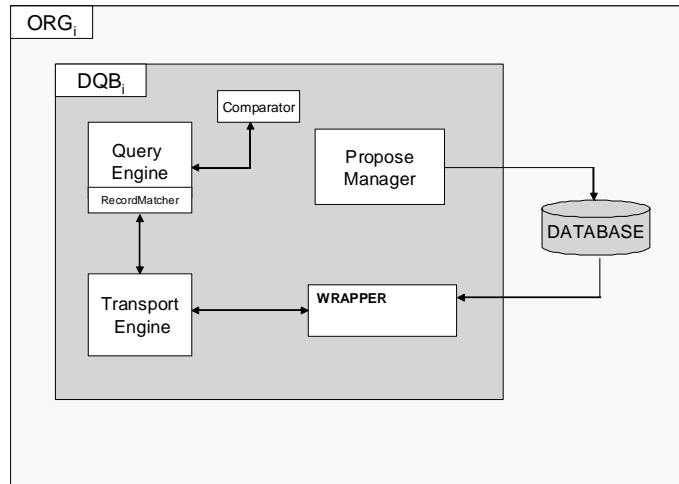


Figure 3: Data Quality Broker modules

**Comparator** : it is used by the Query Engine to compare the quality values returned by the different sources to choose the best one. The comparator can compare quality value vectors according to ranking methods known in literature, such as Simple Additive Weighting (SAW) [9] or Analytical Hierarchy Process (AHP) [14].

## 6.2 Implementation

The Data Quality Broker is implemented as a set of web services peerly interacting with each other and deployed on each organization.

The communication functionality performed by the Transport Engine module, are implemented by using Tomcat 3.2 Apache Jakarta as web server, Apache Soap 2.0 as communication protocol and Apache Xerces XML Parser to parse exchanged XML messages.

The querying functionalities performed by the Query Engine module are based on the XML technologies chosen for the data model and the query language. Specifically, The  $\mathbf{D}^2\mathbf{Q}$  model is coded as XML Schemas and is queried by XQuery extended with accessors to quality data. IPSI-XQ is used as query engine. In order to manage parsing of XQuery queries, a parser has been generated by using the JavaCC (Java Compiler Compiler) tools.

The Record Matcher, the Query Engine and the Transport Engine modules of the Data Quality Broker have been implemented in Java, for a total of more than 12.500 lines of Java code.

## 7 Experiments

In this Section, we show the results of the experiments we performed in order to consider the effectiveness of the *DaQuinCIS* approach in improving quality of data. We first show the experimental methodology that we adopted in Section 7.1; then, in Sections 7.2 and 7.3, we show respectively quality improvement experiments and performance experiments.

### 7.1 Experimental Methodology

We performed two types of experiments. The first set shows the effectiveness of the *DaQuinCIS* system to improve data quality. The second set shows the performance features of *DaQuinCIS*.

We used two real data sets. These data sets are copies of the data sets used in the “Services to Businesses” project, cited in the introduction.

Each data set is 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: (i) features characterizing the business, including one or more identifiers, headquarter and branches addresses, legal form, main economic activity, number of employees and contractors, information about the owners or partners; and (ii) milestone dates, including date of business start-up and (possible) date of cessation.

As far as quality improvement experiments, we have associated quality values to the INPS and CoC databases. Specifically, we have associated accuracy and currency quality values to each field value. Accuracy values are calculated by comparison with reference dictionaries for each field type. As far as currency values, last update timestamps were already associated to data values in the two databases. We have calculated the degree of overlapping of the two databases that is equal to about 670000 records.

As far as performance experiments, a cooperative 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 interact each other using the SOAP protocol.

## 7.2 Quality Improvement Experiments

The first quality improvement experiment considers quality improvement in both databases resulting from the query-time improvement performed by the *DaQuinCIS* system. Quality improvement is measured as *Number of Improved Tuples*. We consider different copies of the experimental databases due to updates of a fixed set of fields composing an address. The frequency of changes of addresses in the two databases has been estimated to be 5000 tuples each 7 days. An average number of queries performed on addresses has been estimated to be 3000. The average size of query results is about 5000 tuples. In Figure 4, the Number of Improved Tuples is drawn with respect to time. Each seven days there is a step because, for the sake of simplicity, we have considered updates at the beginning of each 7 days period. The *DaQuinCIS* curve is depicted as a full line. Moreover, a dashed line represents the improvement in both databases without the *DaQuinCIS* system, in a *standard* system, i.e. a system that does not perform any kind of quality improvement. This latter is a curve consisting of different steps because no improvement is performed between two updates. Instead, in the *DaQuinCIS* system, the curve within a period has a growing trend due to improvement feedbacks at query time. At the end of each period, the *DaQuinCIS* curve reaches a point that is not far from the possible maximum improvement, corresponding to a complete update propagation. The second quality improvement experiment shows the earn of the *DaQuinCIS* system performing query time quality improvement plus periodical record matching with respect to the standard system. In Figure 5, the filled area shows the actual earn in terms of quality improvement obtained by the full application of the *DaQuinCIS* improvement approach. The third quality improvement experiment shows the detail of the behavior of the *DaQuinCIS* query time improvement when considering a single period and varying the number of performed queries and the size of the result. This experiment does not take into account the values of such parameters in the two real databases; instead, it makes them vary in order to study the system reaction. In Figure 6, if fixing the result size to 5000 tuples, increasing the number of queries leads to an improvement of quality. Also, if fixing the number of queries to 200, increasing the result size leads also to an improvement of quality.

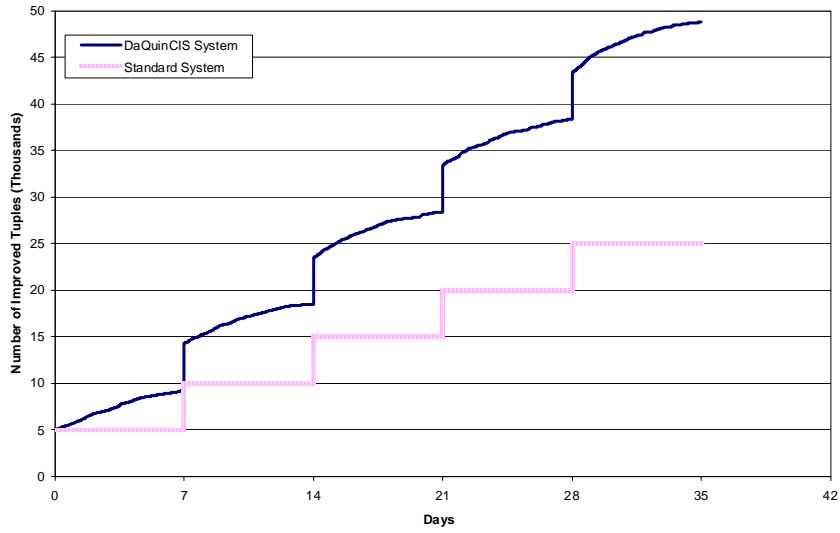


Figure 4: The *DaQuinCIS* improvement at query time compared to a system that does not perform query-time improvement

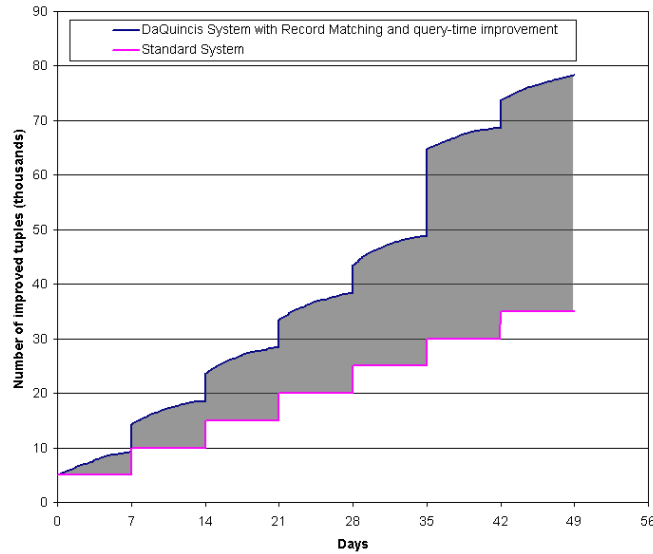


Figure 5: The overall *DaQuinCIS* improvement approach wrt a standard system

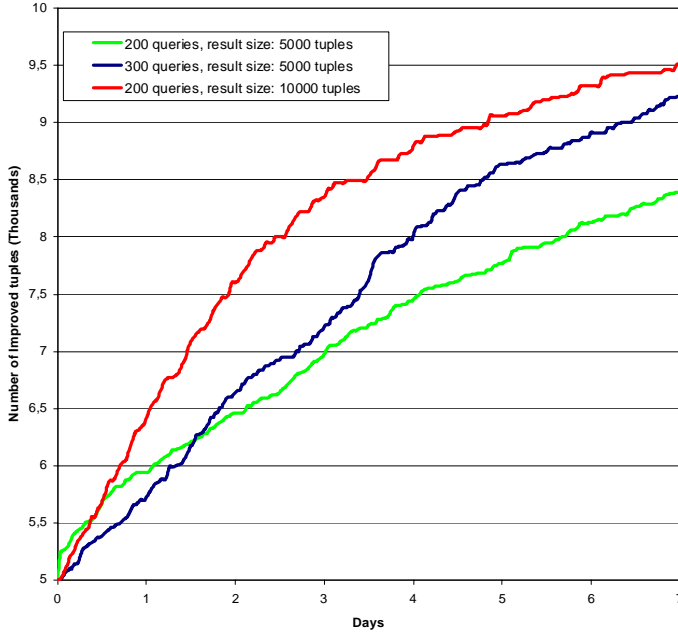


Figure 6: The *DaQuinCIS* improvement at query time within a single period while varying query size and query number

### 7.3 Performance Experiments

The first performance experiment shows the time overhead of the *DaQuinCIS* system with respect to a cooperative system that perform query answering without any quality improvement, called *standard* in the following. More specifically, we draw a *normalized transaction time*, consisting of the fraction:

$$\frac{DaQuinCISElaborationTime - StandardElaborationtime}{StandardElaborationtime}$$

The elaboration time is the time required by the system to answer a query, minus (i) the time for issuing the query and (ii) the time for gathering the result from different data sources; such two times that can be considered comparable for the *DaQuinCIS* system and the standard system. The normalized transaction time is drawn when varying the degree of overlapping of data sources. The overlapping degree is an important feature of the *DaQuinCIS* system. Indeed, the *DaQuinCIS* system accomplishes its functionalities in contexts where data sources overlap and such an overlapping can be exploited to improve the quality of data. The Figure 7 shows how the normalized transaction time varies in dependance on the percentage of data sources overlapping with two fixed query result sizes, namely  $q_1=1000$  tuples,  $q_2=5000$  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 an answer to the queries, though the system can have a larger number of sources. The Figure 7 shows the actual time overhead of the *DaQuinCIS* systems with respect to a standard system. The *DaQuinCIS* system has an acceptable time overhead. The worst depicted case is for the query result size  $q_2=5000$  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 8). For a fixed degree of overlapping, namely 15%, we draw the normalized transaction time for three different numbers of overlapping organizations, namely  $n_1=3$ ,  $n_2=4$  and  $n_3=5$ . This experiment shows the behavior of the *DaQuinCIS* system when increasing the

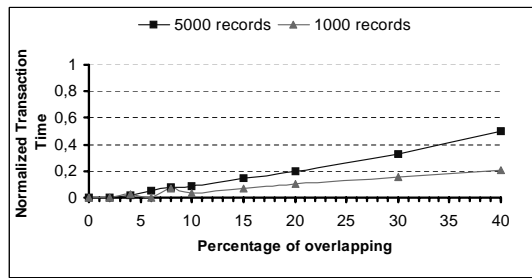


Figure 7: Normalized transaction time wrt percentage of overlapping data sources

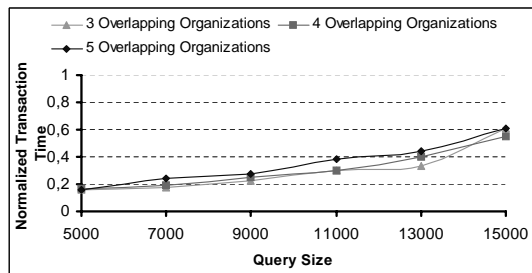


Figure 8: Normalized transaction time wrt query sizes

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

## References

- [1] C. Batini and M. Mecella. Enabling Italian *e*-Government Through a Cooperative Architecture. *IEEE Computer*, 34(2), 2001.
- [2] P. Bertolazzi, L. De Santis, and M. Scannapieco. Automatic Record Matching in Cooperative Information Systems. In *Proceedings of the ICDT'03 International Workshop on Data Quality in Cooperative Information Systems (DQCIS'03)*, Siena, Italy, 2003.
- [3] M. Bertoletti, P. Missier, M. Scannapieco, P. Aimetti, and C. Batini. Improving Government-to-Business Relationships through Data Reconciliation and Process Re-engineering. In Richard Wang, editor, *Advances in Management Information Systems-Information Quality Monograph (AMIS-IQ) Monograph*. Sharpe, M.E., to appear, 2003. Shorter version also in ICIQ 2002.
- [4] C. Cappiello, C. Francalanci, B. Pernici, P. Plebani, and M. Scannapieco. Data Quality Assurance in Cooperative Information Systems: a Multi-dimension Quality Certificate. In *Proceedings of the ICDT'03 International Workshop on Data Quality in Cooperative Information Systems (DQCIS'03)*, Siena, Italy, 2003.
- [5] L. De Santis, M. Scannapieco, and T. Catarci. Trusting Data Quality in Cooperative Information Systems. In *Proceedings of the 11th International Conference on Cooperative Information Systems (CoopIS'03)*, Catania, Italy, 2003.
- [6] S. J. Axford H. B. Newcombe, J. M. Kennedy and A. P.F James. Automatic Linkage of Vital Records. *Science*, 130, 1959.

- [7] M.A. Hernandez and S.J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Journal of Data Mining and Knowledge Discovery*, 1(2), 1998.
- [8] M.A. Hernandez and S.J. Stolfo. The Merge/Purge Problem for Large Databases. In *Proceedings of ACM Special Interest Group on Management Of Data International Conference (SIGMOD 1995)*, San Jose, California, 1995.
- [9] C. Hwang and K. Yoon. *Multiple Attribute Decision Making*. Lectures Notes in Economics and Mathematical Systems-Springer Verlag 186, 1981.
- [10] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the 21st ACM Symposium on Principles of Database Systems (PODS 2002)*, Madison, Wisconsin, USA, 2002.
- [11] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Doklady Akademii Nauk SSSR*, 10(8), 1966.
- [12] C. Marchetti, M. Mecella, M. Scannapieco, and A. Virgillito. Enabling Data Quality Notification in Cooperative Information Systems through a Web-service based Architecture (short paper). In *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE'03)*, Roma, Italy, 2003.
- [13] M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, and C. Batini. Managing Data Quality in Cooperative Information Systems. *Journal of Data Semantics*, to appear, 2003. Shorter version also appeared in CoopIS 2002.
- [14] T.L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980.
- [15] M. Scannapieco, B. Pernici, and E.M. Pierce. IP-UML: A Methodology for Quality Improvement based on IP-MAP and UML. In Richard Wang, editor, *Advances in Management Information Systems-Information Quality Monograph (AMIS-IQ) Monograph*. Sharpe, M.E., to appear, 2003.
- [16] M. Scannapieco, A. Virgillito, M. Marchetti, M. Mecella, and R. Baldoni. The DaQuin-CIS architecture: a platform for exchanging and improving data quality in Cooperative Information Systems. *Information Systems*, to appear, 2003.
- [17] J.D. Ullman. Information Integration Using Logical Views. In *Proceedings of the 6th International Conference on Database Theory (ICDT '97)*, Delphi, Greece, 1997.
- [18] R.Y. Wang and S. Madnick. The Inter-Database Instance Identification Problem in Integrating Autonomous Systems. In *Proceedings of the 5th International Conference on Data Engineering (ICDE'89)*, Los Angeles, California, USA, 1989.