Managing Data Quality in Cooperative Information Systems

Massimo Mecella[1], Monica Scannapieco[1,2], Antonino Virgillito[1], Roberto Baldoni[1], Tiziana Catarci[1], and Carlo Batini[3]

[1] Università di Roma "La Sapienza", DIS
{mecella,monscan,virgi,baldoni,catarci}@dis.uniroma1.it
[2] Consiglio Nazionale delle Ricerche, IASI
[3] Università di Milano "Bicocca", DISCo
batini@disco.unimib.it

Abstract. Current approaches to the development of cooperative information systems are based on services to be offered by cooperating organizations, and on the opportunity of building coordinators and brokers on top of such services. The quality of data exchanged and provided by different services hampers such approaches, as data of low quality can spread all over the cooperative system. At the same time, improvement can be based on comparing data, correcting them and disseminating high quality data. In this paper, a service-based framework for managing data quality in cooperative information systems is presented. An XML-based model for data and quality data is proposed, and the design of a broker, which selects the best available data from different services, is presented. Such a broker also supports the improvement of data based on feedbacks to source services.

1 Introduction

A *Cooperative Information System (CIS)* is a large scale information system that interconnects various systems of different and autonomous organizations, geographically distributed and sharing common objectives. Among the different resources that are shared by organizations, data are fundamental; in real world scenarios, an organization $\mathcal{A}$ may not request data from an organization $\mathcal{B}$ if it does not "trust" $\mathcal{B}$ data, i.e., if $\mathcal{A}$ does not know that the quality of the data that $\mathcal{B}$ can provide is high. As an example, in an *e*-Government scenario in which public administrations cooperate in order to fulfill service requests from citizens and enterprises [1], administrations very often prefer asking citizens for data, rather than other administrations that have stored the same data, because the quality of such data is not known. Therefore, lack of cooperation may occur due to lack of quality certification.

Uncertified quality can also cause a deterioration of the data quality inside single organizations. If organizations exchange data without knowing their actual quality, it may happen that data of low quality spread all over the CIS.

R. Meersman, Z. Tari (Eds.): CoopIS/DOA/ODBASE 2002, LNCS 2519, pp. 486–502, 2002.
© Springer-Verlag Berlin Heidelberg 2002

On the other hand, CIS's are characterized by high data replication, i.e., different copies of the same data are stored by different organizations. From a data quality perspective this is a great opportunity: improvement actions can be carried out on the basis of comparisons among different copies, in order either to select the most appropriate one or to reconcile available copies, thus producing a new improved copy to be notified to all involved organizations.

CIS's designed according to a service-based approach [2] consider cooperation among different organizations to be obtained by sharing and integrating services across networks; such services, commonly referred to as *e*-Services and Web-Services [3], are exported by different organizations as well defined operations that allow users and applications to access and perform tasks offered by back-end business applications.

In this paper, we propose a service-based framework and an overall architecture for managing data quality in CIS's. The architecture aims at avoiding dissemination of low qualified data through the CIS, by providing a support for data quality diffusion and improvement. At the best of our knowledge, this is a novel contribution in the information quality area, that aims at integrating, in the specific context of CIS, both modeling and architectural issues. To enforce this vision, our work, beside presenting the general architecture, focuses on two specific elements of the problem, that we consider of primary importance. More specifically:

- ❒ we first face the problem of lack of quality certification by proposing a model for each organization to export data with associated quality information. The model is XML-based in order to address interoperability issues existing in cooperative information systems;
- ❒ then, we present the distributed design of a single architectural service, based on the model cited above, namely the Data Quality Broker, which allows each organization involved in the CIS to retrieve data specifying their quality requirements. The service offers only data that satisfies the given requirements and notifies organizations about the highest quality values found within the CIS. The design of the distributed service takes into account reliable communication issues and shows the feasibility of our approach in practical scenarios.

The structure of the paper is as follows. In Section 2, related research work is discussed. In Section 3, a service-based framework for Cooperative Information Systems is proposed. On the basis of such a framework, in Section 4, an architecture specifically addressing quality related issues is described. In Section 5, a model for both the exchanged data and their quality is presented. In Section 6, the distributed design of the Data Quality Broker service is described. Finally, Section 7 concludes the paper by drawing future work.

2 Related Work

Data Quality has been traditionally investigated in the context of single information systems; only recently, a methodological framework for data quality in

cooperative systems has been proposed, consisting of five phases (i.e., definition, measurement, exchange, analysis and improvement) [4].

In cooperative scenarios, the main data quality issues regard: *(i)* assessment of the quality of the data owned by each organization; *(ii)* methods and techniques for exchanging quality information; *(iii)* improvement of quality within each cooperating organization; and *(iv)* heterogeneity, due to the presence of different organizations, in general with different data semantics.

For the assessment *(i)* and the heterogeneity *(iv)* issues, some of the results already achieved for traditional systems can be borrowed, e.g., [5,6]. Methods and techniques for exchanging quality information *(ii)* have been only partially addressed in the literature. When considering the issue of exchanging data and the associated quality, a model to export data and quality data needs to be defined. Some conceptual models to associate quality information to data have been proposed: an extension of the Entity-Relationship model [7], and a data warehouse conceptual model with quality features described through the Description Logic formalism [6]. Both models are thought for a specific purpose: the former to introduce quality elements in relational database design; the latter to introduce quality elements in the data warehouse design. Whereas, in the present paper the aim is to enable quality exchanging in a generic CIS, independently of the specific data model and system architecture.

In [8], the problem of the quality of web-available information has been faced in order to select data with high quality coming from distinct sources: every source has to evaluate some pre-defined data quality parameters, and to make their values available through the exposition of meta-data. Our proposal is different as we propose an ad-hoc service that brokers data requests and replies on the basis of data quality information. Moreover, we also take into account improvement features *(iii)* that are not considered in [8].

3 The Framework

In current business scenarios, organizations need to cooperate in order to offer services to their customers and partners. Organizations that cooperate have business links (i.e., relationships, exchanged documents, resources, knowledge, etc.) connecting each other. Specifically, organizations exploit business services (e.g., they exchange data or require services to be carried out) on the basis of business links, and therefore the network of organizations and business links constitutes a cooperative business system.

As an example, a supply chain, in which some enterprises offer basic products and some others assemble them in order to deliver final products to customers, is a cooperative business system. As another example, a set of public administrations which need to exchange information about citizens and their health state in order to provide social aids, is a cooperative business system derived from the Italian *e*-Government scenario [1].

A cooperative business system exists independently of the presence of a software infrastructure supporting electronic data exchange and service provisioning.

Indeed cooperative information systems are software systems supporting cooperative business systems; in the remaining of this paper, the following definition of CIS is considered:

A cooperative information system *is formed by a set of organizations* $\{ Org_1, \ldots, Org_n \}$ *which cooperate through a communication software infrastructure* $\mathbb{N}$*, which may provide software services, referred to as infrastructure services (IS's), to organizations as wells as reliable connectivity. Each organization* Org_i *is connected to* $\mathbb{N}$ *through a gateway* G_i*, on which application services (AS's) offered by* Org_i *to other organizations are deployed. We denote as* $\mathcal{AS}_{j,i}$ *the* j*-th application service offered by* Org_i*.*

The difference between application and infrastructure services is that the latter can be designed independently of organizations. Application services can perform different operations, such as initiating complex transactions on backend systems, providing access to data, etc. In the present work we only consider *read-only access* services, that is application services returning application data stored inside organizations without modifying them. We will assume the following definition of application service:

A generic application service $\mathcal{AS}_{j,i}$ *offered by a cooperating organization* Org_i *is a set of operations* $\{s_1, ..., s_n\}$*, each one specified by a signature of the following form:*

$$s_i(p_1, ..., p_n) \rightarrow \{\mathcal{O}_1, ..., \mathcal{O}_n\}$$

where $p_1, ..., p_n$ *is the list of parameters and* $\{\mathcal{O}_1, ..., \mathcal{O}_n\}$ *is the set of data items returned by* s_i*.*

The $\mathcal{O}_i$'s returned by application services are expressed as XML documents that convey not only application data items, but also data about the quality of such data items (see Section 5).

4 An Architecture for Data Quality

A typical feature of CIS's is the high degree of data replicated in different organizations. As an example, in an *e*-Government scenario, the personal data of a citizen are stored by almost all the administrations. On the basis of the proposed definition of CIS, more than one organization can implement the same application service. Therefore, according to our assumption of considering only access data services, several organizations can provide the same data though with different quality levels. Any requester of data may want to have the data with the highest quality level, among the provided ones. Thus only the highest quality data are returned to the requester, limiting the dissemination of low quality data. Moreover, the comparison of the gathered data values can be used to enforce a general improvement of data quality in all the organizations.

In the context of the DaQuinCIS project[1], we propose an architecture for the management of data quality in CIS's; such an architecture allows the diffusion of data and related quality and exploits data replication to improve the overall quality of cooperative data. According to the logical model of a CIS presented in the previous section, we need to define both a model for the organizations to exchange data and data quality data and a set of infrastructure services that realize quality management functions.

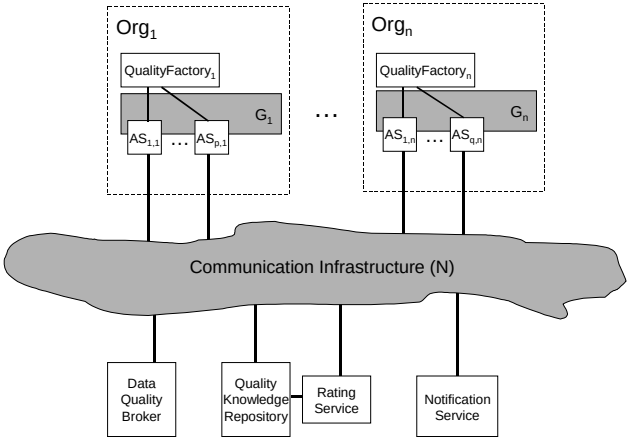


Fig. 1. An architecture for data quality diffusion and improvement

The model for data quality we propose in this paper is called *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. The D^2Q model is described in Section 5.

In order to produce data and quality data according to the D^2Q model, each organization holds a **Quality Factory** that is responsible for evaluating the quality of its own data. The overall architecture is depicted in Figure 1. In the following we give a description of each element:

- **Data Quality Broker**: it is the core of the architecture. It performs, on behalf of a requesting organization, a data request on all the AS's, also specifying a set of quality requirements that the desired data have to satisfy (*quality brokering function*). Different copies of the same data received as responses to the request are reconciled and a best-quality value is selected

[1] "DaQuinCIS - Methodologies and Tools for Data Quality in Cooperative Information Systems" is an Italian research project carried out by Università di Roma "La Sapienza", Università di Milano "Bicocca" and Politecnico di Milano (`http://www.dis.uniroma1.it/~dq/`).

and proposed to organizations, that can choose to discard their data and adopt higher quality ones (*quality improvement function*). Quality brokering and improvement are described in Section 6. If the requirements specified in the request cannot be satisfied, then the broker initiates a negotiation with the requester that can optionally weaken the constraints on the desired data.

- ❒ **Quality Knowledge Repository**: it consists of a knowledge base used by the other components in order to perform their functions. For example, it maintains *(i)* the interschema knowledge representing the relationships among schemas that allow to determine intensional and extensional equivalence of data in different organizations [10], and *(ii)* historical quality knowledge, including statistics related to data quality ensured by organizations in the past, that is also used to realize a rating service for source reliability.
- ❒ **Notification Service**: it is a publish/subscribe engine used as a general message bus between components and/or organizations. More specifically, it allows quality-based subscriptions for organizations to be notified for quality changes in data. For example, an organization may want to be notified if the quality of a data it uses degrades below a certain acceptable threshold, or when high quality data are available.
- ❒ **Rating Service**: it associates trust values to each data source in the CIS. These are used by the Data Quality Broker to determine the reliability of the quality evaluation made by organizations. Trust values are dynamically updated from the statistics from the Quality Knowledge Repository and take also into account the current availability of the data source.

The detailed design of such components is currently under investigation [9, 11]; in this paper, we only focus on the architectural design of the data quality broker, which is detailed in Section 6.

5 The D^2Q Model

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. Exported data and quality data can be accessed by other organizations by means of application service operations that each cooperating organization makes available to the others. The model for exporting data and quality data is referred to as *Data and Data Quality (D^2Q) model*. In this section, we first introduce the data quality dimensions used in this paper (Section 5.1), then we describe the D^2Q model with respect to the data features (Section 5.2) and the quality features (Section 5.3).

5.1 Data Quality Dimensions

Data quality dimensions are properties of data such as correctness or degree of updating. The data quality dimensions used in this work concern only data values; instead, they do not deal with aspects concerning quality of logical schema and data format [12].

In this section, we propose and outline some data quality dimensions to be used in CIS's, stemming from real requirements of CIS's scenarios that we experienced [1]. The reader should refer to [9] for complete definitions, examples and possible evaluation methods related to each of them.

In the following, the general concept of *schema element* is used, corresponding, for instance, to an entity in an Entity-Relationship schema or to a class in a Unified Modeling Language diagram. We define:

- **Accuracy**. It is the distance between *v* and *v'*, being *v'* the value considered as correct.
- **Completeness**. It is the degree to which values of a schema element are present in the schema element instance.
- **Currency**. The currency dimension refers only to data values that may vary in time; as an example, values of `Address` may vary in time, whereas `DateOfBirth` can be considered invariant. Therefore, currency can be defined as the "age" of a value. Namely, currency is the distance between the instant when a value changes in the real world and the instant when the value itself is modified in the information system.
- **Internal Consistency**. Consistency implies that two or more values do not conflict each other. Internal consistency means that all values being compared in order to evaluate consistency are within a specific instance of a schema element. A semantic rule is a constraint that must hold among values of attributes of a schema element, depending on the application domain modeled by the schema element. Then, internal consistency can be defined as the degree to which the values of the attributes of an instance of a schema element satisfy the specific set of semantic rules defined on the schema element.

5.2 Data Model

The D^2Q model is inspired by the data model underlying XML-QL [13]. A database view of XML is adopted: an XML Document is a set of data items, and a Document Type Definition (DTD) is the schema of such data items, consisting of *data* and *quality classes*. In particular, a D^2Q XML document contains both application data, in the form of a D^2Q *data graph*, and the related data quality values, in the form of four D^2Q *quality graphs*, one for each quality dimension introduced in Section 5.1. Specifically, nodes of the D^2Q data graph are linked to the corresponding ones of the D^2Q quality graphs through links, as shown in Figure 2. Operations offered by the application services return D^2Q *XML documents* as outputs.

A D^2Q XML document corresponds to a set of conceptual data items, which are instances of conceptual schema elements; schema elements are data and quality classes, and instances are data and quality objects. Data classes and objects are straightforwardly represented as D^2Q data graphs, as detailed in the following of this section, and quality classes and objects are represented as D^2Q quality graphs, as detailed in Section 5.3.

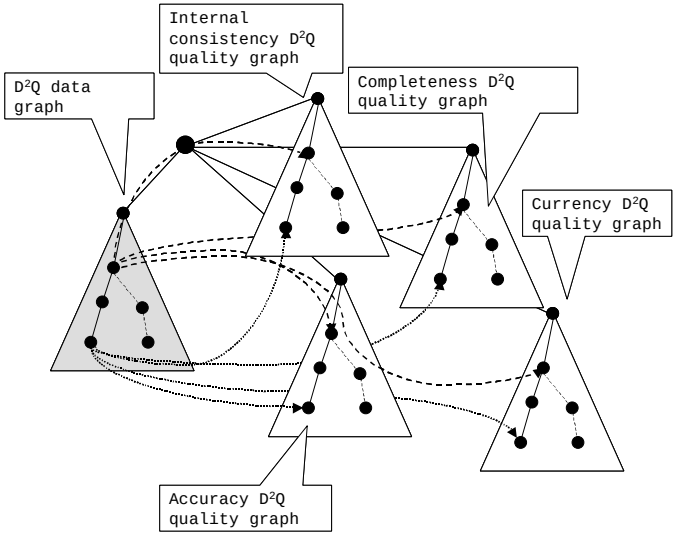


Fig. 2. The generic structure of a D^2Q XML document, returned as result by a service operation

In order to clarify our definition of data class in XML, we preliminary recall a typical definition of data class from ODMG [14].

A data class $\delta\ (\pi_1, \ldots, \pi_n)$ *consists of:*

- *a name* δ*;*
- *a set of properties* $\pi_i = < name_i : type_i >$, $i = 1 \ldots n$, $n \geq 1$, *where* $name_i$ *is the name of the property* π_i *and* $type_i$ *can be:*
  - *either a basic type*[2]*;*
  - *or a data class;*
  - *or a type* `set-of < X >`*, where* `< X >` *can be either a basic type or a data class.*

We define a D^2Q *data graph* as follows:

A D^2Q data graph $\mathbb{G}$ *is a graph with the following features:*

- *a set of nodes* $\mathcal{N}$*; each node* (i) *is identified by an object identifier and* (ii) *is the source of 4 different links to quality objects, each one for a different quality dimension. A link is a pair attribute-value, in which attribute represents the specific quality dimension for the element tag and value is an* `IDREF` *link*[3]*;*

[2] Basic types are the ones provided by the most common programming languages and SQL, that is `Integer`, `Real`, `Boolean`, `String`, `Date`, `Time`, `Interval`, `Currency`, `Any`.

[3] The use of links will be further explained in Section 5.3, when quality graphs are introduced.

- *a set of edges* $\mathcal{E} \subset \mathcal{N} \times \mathcal{N}$*; each edge is labeled by a string, which represents an element tag of an XML document;*
- *a single root node* $\mathcal{R}$*;*
- *a set of leaves; leaves are nodes that* (i) *are not identified and* (ii) *are labeled by strings, which represent element tag values, i.e., the values of the element tags labeling edges to them.*

Data class instances can be represented as D^2Q data graphs, according to the following rules.

Let $\delta\ (\pi_1, \ldots, \pi_n)$ *be a data class with n properties, and let* $\mathcal{O}$ *be a data object, i.e., an instance of the data class. Such an instance is represented by a* D^2Q *data graph* $\mathbb{G}$ *as follows:*

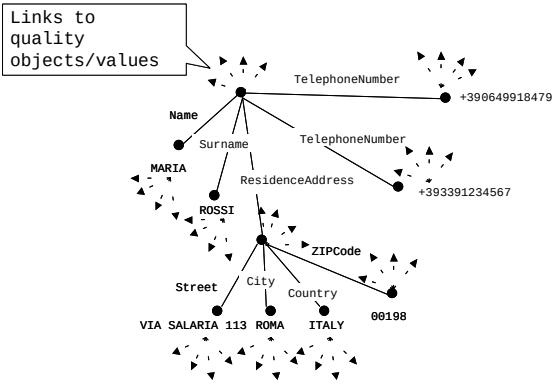
- *The root* $\mathcal{R}$ *of* $\mathbb{G}$ *is labeled with the object identifier of the instance* $\mathcal{O}$*.*
- *For each* $\pi_i = < name_i : type_i >$ *the following rules hold:*
  - *if* $type_i$ *is a basic type, then* $\mathcal{R}$ *is connected to a leaf* lv_i *by the edge* $< \mathcal{R}, lv_i >$*; the edge is labeled with* $name_i$ *and the leaf* lv_i *is labeled with the property value* $\mathcal{O}.name_i$*;*
  - *if* $type_i$ *is a data class, then* $\mathcal{R}$ *is connected to the* D^2Q *data graph which represents the property value* $\mathcal{O}' = \mathcal{O}.name_i$ *by an edge labeled with* $name_i$*;*
  - *if* $type_i$ *is a* `set-of < X >`*, then:*
    - *let* C *be the cardinality of* $\mathcal{O}.name_i$*;* $\mathcal{R}$ *is connected to* C *elements as it follows: if* (i) `< X >` *is a basic type, then the elements are leaves (each of them labeled with a property value of the set); otherwise if* (ii) `< X >` *is a data class, then the elements are* D^2Q *data graphs, each of them representing a data object of the set;*
    - *edges connecting the root to the elements are all labeled with* $name_i$*.*

In Figure 3(a), a D^2Q data graph is shown: an object instance `Maria Rossi` of the data class `Citizen` is considered. The data class has `Name` and `Surname` as properties of basic types, a property of type `set-of < TelephoneNumber >` and another property of data class type `ResidenceAddress`; the data class `ResidenceAddress` has all properties of basic types.

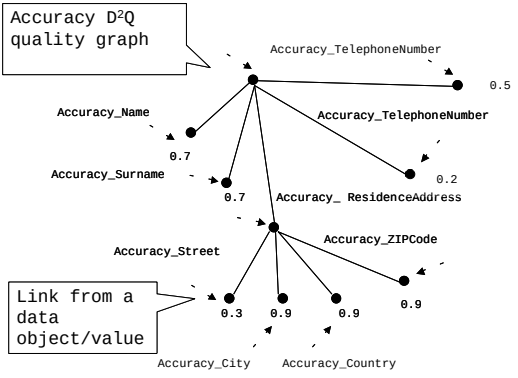
5.3 Quality Model

So far the data portion of the D^2Q model has been described. However, organizations export XML documents containing not only data objects, but also quality data concerning the four dimensions introduced in Section 5.1.

Quality data are represented as graphs, too; they correspond to a set of conceptual quality data items, which are instances of conceptual quality schema elements; quality schema elements are referred to as *quality classes* and instances as *quality objects*. A quality class models a specific quality dimension for a specific data class: the property values of a quality object represent the quality dimension values of the property values of a data object. Therefore, each data object (i.e.,



(a) D^2Q data graph



(b) Accuracy D^2Q quality graph

Fig. 3. An example

node) and value (i.e., leaf) of a D^2Q data graph is linked to respectively four quality objects and values.

Let $\delta\ (\pi_1,\ \ldots,\ \pi_n)$ *be a data class. A quality class* $\delta^D\ (\pi_1^D,\ \ldots,\ \pi_n^D)$ *consists of:*

- ❒ *a name* δ^D, *with* $D \in \{$ `Accuracy, Completeness, Currency, InternalConsistency` $\}$;
- ❒ *a set of tuples* $\pi_i^D = < name_i^D : type_i^D >$, $i = 1 \ldots n$, $n \geq 1$,

where:

- δ^D *is associated to* δ *by a one-to-one relationship and corresponds to the quality dimension* D *evaluated for* δ*;*
- π_i^D *is associated to* π_i *of* δ *by a one-to-one relationship, and corresponds to the quality dimension* D *evaluated for* π_i*;*
- $type_i^D$ *is either a basic type or a quality class or a* `set-of` *type, according to the structure of the data class* δ*.*

In order to represent quality objects, we define a D^2Q *quality graph* as follows:

A D^2Q quality graph $\mathbb{G}^D$ *is a* D^2Q *data graph with the following additional features:*

- *no node nor leaf is linked to any other element;*
- *labels of edges are strings of the form* `D_name` *(e.g.,* `Accuracy_Citizen`*);*
- *labels of leaves are strings representing quality values;*
- *leaves are identified by object identifiers.*

A quality class instance can be straightforwardly represented as a D^2Q quality graph, on the basis on rules analogous to the ones previously presented for data objects and D^2Q data graphs. As an example, in Figure 3(b), a D^2Q quality graph concerning accuracy is shown, and links are highlighted; for instance, the accuracy of `Maria` is `0.7`.

Service operations return a result document which consists of D^2Q graphs. Specifically, for each data class instance there is a D^2Q data graph linked to four D^2Q quality graphs, expressing the quality of the data objects for each dimension introduced in Section 5.1.

Let $\{ \mathcal{O}_1, \ldots, \mathcal{O}_m \}$ *be a set of* m *objects which are instances of the same data class* δ*; a* D^2Q XML document *is a graph consisting of:*

- *a root node* $\mathcal{ROOT}$*;*
- m D^2Q *data graph* $\mathcal{G}_i$*,* $i = 1 \ldots m$*, each of them representing the data objects* $\mathcal{O}_i$*;*
- $4 * m$ D^2Q *quality graph* $\mathcal{G}_i^D$*,* $i = 1 \ldots m$*, each of them representing the quality graph related to* $\mathcal{O}_i$ *concerning the quality dimension* D*;*
- $\mathcal{ROOT}$ *is connected to the* m D^2Q *data graphs by edges labeled with the name of the data class, i.e.,* δ*;*
- *for each quality dimension* D*,* $\mathcal{ROOT}$ *is connected to the* m D^2Q *quality graph* $\mathcal{G}_i^D$ *by edges labeled with the name of the quality class, i.e.,* δ^D*.*

The model proposed in this work adopts several graphs instead of embedding metadata within the data graph. Such a decision increases the document size, but on the other hand allows a modular and "fit-for-all" design: *(i)* extending the model to new dimensions is straightforward, as it requires to define the new dimension quality graph, and *(ii)* specific applications, requiring only some dimension values, will adopt only the appropriate subset of the graphs.

6 The Data Quality Broker

In this section, we present a distributed implementation of the Data Quality Broker component (DQB). For some issues concerning the high level design of this component, as well as examples of its possible usage in real scenarios, the reader should also refer to [15].

As mentioned in Section 4, the DQB service provides the following features:

- ❒ **Quality brokering function:** an organization can invoke DQB specifying a data request constrained by a set of data quality requirements. The data request is performed by invoking an operation s. DQB discovers (through the Quality Knowledge Repository) all application services in the CIS offering s or operations equivalent to s[4], invokes them, gathers all responses and then returns only data items satisfying the requirements.
- ❒ **Quality improvement function:** DQB notifies organizations with low quality data and proposes them the highest quality value obtained in the previous step. Organizations may choose to adopt such a highest quality data.

The idea behind the brokering function is that DQB filters the normal interaction that a client has with organizations in the CIS, adding the possibility to discover and invoke equivalent services in different organizations and to execute a quality-based filtering of the results obtained. The improvement function is a feedback mechanism that uses the results of the brokering to propagate highest quality data.

The design of DQB is presented by specifying how the above functions are provided to organizations by each copy of the service and implemented through distributed protocols. A distributed service is inherently more scalable and robust than a service implemented as a centralized component, since request load is automatically shared among all the components implementing DQB and the fault of a single component does not impact on the availability of the whole service. These features make this choice more appropriate in the CIS context.

Details on how to perform quality comparison and how to determine the highest quality value are abstracted in this context, since we assume they are provided from other architectural components.

6.1 Preliminary Assumptions

In order to model the communication environment of the CIS we make the following assumptions:

- ❒ both application and infrastructure services are implemented by one or more software components. Such software components can fail by *crashing*, thus causing the *down* of the service;

[4] In the context of this paper, we simply consider an intuitive definition of operational equivalence, i.e., two operations are equivalent if they return equivalent data, as defined in the Quality Knowledge Repository.

- the communication infrastructure $\mathbb{N}$ is abstracted as an *asynchronous distributed system* [16]: communication links provided by $\mathbb{N}$ are reliable (i.e., each message sent by a non-down component is eventually delivered to a non-down recipient) but message transfer delay is unpredictable. Moreover, a service can be arbitrarily slow down due to unpredictable service workload.

On the basis of such assumptions, it is impossible to distinguish a down service from one that is up but extremely slow [17]; therefore, in order to guarantee service termination, an additional assumption is required: only a minority of DQB software components can crash at the same time.

6.2 Service Design

Some preliminary definitions are required before specifying the service. We define a quality requirement as a constraint on the value of a specific dimension over a property of a data object, represented as a D^2Q data graph:

A data quality requirement *is a set* $qr = \{ p, D, op, v \}$*, where* p *is the name of a property,* D *is a data quality dimension defined in the* D^2Q *model,* op *can be* $=, <, >, \leq$ *or* $\geq$*, and* v *is a value defined in the domain of the data quality dimension* D*.*

If $eval(D, \mathcal{O})$ is a function that assesses the quality of D for the data object $\mathcal{O}$, we say that a data object $\mathcal{O}$ *satisfies* a data quality requirement $qr = \{ p, D, op, v \}$ if $\mathcal{O}$ "contains" a property that corresponds to p and $eval(D, \mathcal{O})$ op v is true.

The input to the primitive *read*, which will be defined in the following, is a set of quality requirements $\mathcal{QR} = \{ qr_1, \ldots, qr_n \}$; $\mathcal{O}$ satisfies $\mathcal{QR}$ when it satisfies each $qr_i \in \mathcal{QR}$.

The functions of DQB are provided by two primitives:

- $read(s, \mathcal{QR})$: it invokes the operation s upon all the AS's that implement it or an equivalent operation, and returns only the data that satisfy the set of quality requirements $\mathcal{QR}$. It represents the way a software system belonging to an organization can access the DQB service;
- $propose(\mathcal{O})$: it proposes the data object $\mathcal{O}$ to organizations. It is the mechanism that DQB uses in order to provide organizations with quality feedbacks.

The DQB service is invoked by a software system inside an organization Org_i by making a call to the *read* primitive. *read* will return to its caller all (and only) the data objects satisfying all the requirements. Among all the obtained objects, one will be chosen to be proposed to each organization, through the *propose* primitive; the *propose* is used by the DQB service to give a non-invasive feedback upon the quality of a data object, allowing an organization to improve the quality of its data while maintaining its autonomy in the decision.

The implementation of the service relies on a function abstracted as a further primitive realized in cooperation with other architectural components:

❒ *compare*$(\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}, \mathcal{QD})$: it returns the data object with the highest quality in the set $\{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$, with respect to the set of quality dimensions $\mathcal{QD}$.

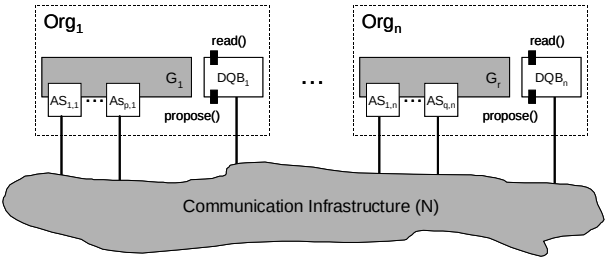


Fig. 4. Deployment of the Data Quality Broker service

The *compare* primitive abstracts the execution of a quality comparison among a set of data instances. The comparison is possible only if the two data objects are instance of equivalent classes, otherwise the returned result is not determined. Here we do not specify the mechanisms and the criteria used for the comparison. These issues are beyond the scope of this paper. We just point out that the comparison have to take into account the trust values provided by the Rating Service, in order to avoid that erroneous evaluations from mistrusted organizations let low quality data propagate.

6.3 DQB Protocols

The DQB service is implemented by a set of identical software components $\{DQB_1, .., DQB_n\}$, where each DQB_i is deployed inside a different organization Org_i and is executed independently of the others (see Figure 4).

Each DQB_i locally implements the primitives defined in the previous section. Moreover, it has access to a *reliable multicast* primitive [18], denoted $RM(m)$, used to send a message m to other DQB_i with specific delivery guarantees. Informally, reliable multicast ensures that each non-crashed component eventually delivers the same set of messages sent by other components. The primitive can be implemented in an asynchronous distributed system with simple deterministic algorithms [18].

We describe in the following the steps of the protocol executed by DQB components when a $read(s, \mathcal{QR})$ is invoked on a DQB_i. For the sake of simplicity, we assume the same operation s as implemented by more than one organization. Real-world cases present equivalent operations returning equivalent data, discovered through the Quality Knowledge Repository. The protocol works as follows (an example of the run of the protocol is depicted in Figure 5):

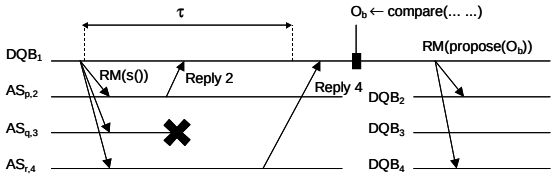


Fig. 5. An example of the Data Quality Broker execution

1. Let Org_i be the invoking organization, DQB_i invokes s on each application service $AS_{j,\ell}$ with $\ell \neq i$, by issuing $RM(s)$ to the group of application services implementing s. Then DQB_i starts a timeout τ.
2. Each $AS_{j,\ell}$ returns to DBQ_i a D^2Q XML document containing objects and their quality.
3. If the timeout τ expires before all the application services have replied, DQB_i further waits only till the majority is reached. This condition eventually will occur as previously supposed.
4. After gathering the replies, DQB_i will return to its caller a D^2Q XML document $\mathcal{Y}$ including the data objects that satisfy the quality requirements.
5. DQB_i applies the *compare* primitive in order to obtain the highest quality data object $\mathcal{O}_b$ among the ones in $\mathcal{Y}$.
6. DQB_i invokes $propose(\mathcal{O}_b)$ on each DQB component by issuing $RM(\ propose(\mathcal{O}_b)\ )$.

Intuitively, the protocol always terminates, exploiting the preliminary condition that at least a majority of components will reply. The reliable multicast primitive, which is used for the communication, also ensures that all the non-crashed DQB components receive the proposal. Note that in order to ensure termination, the service can guarantee only a best-effort semantic, i.e., it might not return all the data which satisfy the requirements. In this case a negotiation can start between DQB and the requester, that can choose to relax the conditions both for quality of data and for protocol parameters, for example specifying less strict quality requirements or a higher timeout value.

7 Conclusions and Future Work

Managing data quality in CIS's merges issues from many research areas of computer science such as databases, software engineering, distributed computing, security, and information systems. This implies that the proposal of integrated solutions is very challenging. In this paper, an overall framework to support data quality management in CIS's has been proposed. Specifically, this framework includes *(i)* a model for data and quality data exported by cooperating organizations and *(ii)* the design of an infrastructure service for brokering and improving quality data.

The complete development of a complex framework for data quality management in CIS's requires the solution of further issues, such as:

- ❒ Application services considered in the work are simple read-only data access services, that export data with the associated quality. More general application services (i.e., encapsulating business logic, able to make updates, etc.) will be considered; the impact of such extensions on the quality of the data they "manage" will be considered.
- ❒ Algorithms for data quality improvement in distributed settings will be also investigated. Given multiple copies of the same data, there are two ways according to which it is possible to engage improvement actions: *(i)* a *reconciliation approach*, consisting of reconciling the differences among the multiple copies into a single representation; *(ii)* a *selecting approach*, that implies the choice of the copy of data with the best quality. Both these approaches have been widely considered in the literature, but in our framework we have the further opportunity to rely on the available quality data in adopting each of them.
- ❒ The "reliability" of cooperating organizations and more generally trust issues also need to be taken into account; as an example, by considering what happens if an organization provides data with a low quality but certifies a high quality for them.

Acknowledgments. This work is supported by MIUR, COFIN 2001 Project "DaQuinCIS – Methodologies and Tools for Data Quality in Cooperative Information Systems" (`http://www.dis.uniroma1.it/~dq/`).

References

1. C. Batini and M. Mecella, "Enabling Italian *e*-Government Through a Cooperative Architecture," *IEEE Computer*, vol. 34, no. 2, 2001.
2. U. Dayal, M. Hsu, and R. Ladin, "Business Process Coordination: State of the Art, Trends and Open Issues," in *Proceedings of the 27th Very Large Databases Conference (VLDB 2001)*, Roma, Italy, 2001.
3. F. Casati, D. Georgakopoulos, and M.C. Shan, Eds., *Proceedings of the 2nd VLDB International Workshop on Technologies for* e-*Services (VLDB-TES 2001)*, Rome, Italy, 2001.
4. P. Bertolazzi and M. Scannapieco, "Introducing Data Quality in a Cooperative Context," in *Proceedings of the 6th International Conference on Information Quality (IQ'01)*, Boston, MA, USA, 2001.
5. H. Galhardas, D. Florescu, D. Shasha, and E. Simon, "An Extensible Framework for Data Cleaning," in *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, San Diego, CA, USA, 2000.
6. M. Jarke, M. Lenzerini, Y. Vassiliou, and Panos Vassiliadis, Eds., *Fundamentals of Data Warehouses*, Springer Verlag, 1999.
7. H.B. Kon, R.Y. Wang and S.E. Madnick, "Data Quality Requirements: Analysis and Modeling," in *Proceedings of the 9th International Conference on Data Engineering (ICDE '93)*, Vienna, Austria, 1993.
8. G. Mihaila, L. Raschid, and M. Vidal, "Querying Quality of Data Metadata," in *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, Valencia, Spain, 1998.

9. M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, and C. Batini, "Architectural Support for Data Quality in Cooperative Information Systems," Technical report of the DaQuinCIS project, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy, 2002.
10. T. Catarci and M. Lenzerini, "Representing and Using Interschema Knowledge in Cooperative Information Systems," *Journal of Intelligent and Cooperative Information Systems*, vol. 2, no. 4, 1993.
11. M. Scannapieco, "Data Quality in Cooperative Information Systems," Doctoral Poster at the 28th Very Large Databases Conference (VLDB 2002), Hong Kong, 2002.
12. T.C. Redman, *Data Quality for the Information Age*, Artech House, 1996.
13. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, "XML-QL: A Query Language for XML," in *Proceedings of the 8th International World Wide Web Conference (WWW8)*, Toronto, Canada, 1999.
14. R.G.G. Cattell and D.K. Barry, Eds., *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, 1997.
15. M. Scannapieco, V. Mirabella, M. Mecella, and C. Batini, "Data Quality in *e*-Business," in *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web: Foundations, Models, Architecture, Engineering and Applications, in conjunction with CAiSE 2002*, Toronto, Ontario, Canada, 2002.
16. F.B. Schneider, "What Goods are Models and What Models are Good?," in *Distributed Systems*, S. Mullender, Ed. ACM Press, 1994.
17. M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, vol. 32, no. 2, 1985.
18. V. Hadzilacos and S. Toueg, "Fault-Tolerant Broadcasts and Related Problems," in *Distributed Systems*, S. Mullender, Ed. ACM Press, 1994.