The DaQuinCIS Architecture: a Platform for Exchanging and Improving Data Quality in Cooperative Information Systems ⋆

Monica Scannapieco * Antonino Virgillito Carlo Marchetti
Massimo Mecella Roberto Baldoni

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113 - 00198 Roma, Italy

Abstract

In cooperative information systems, the quality of data exchanged and provided by different data sources is extremely important. A lack of attention to data quality can imply data of low quality to spread all over the cooperative system. At the same time, improvement can be based on comparing data, correcting them and thus disseminating high quality data. In this paper, we present an architecture for managing data quality in cooperative information systems, by focusing on two specific modules, the Data Quality Broker and the Quality Notification Service. The Data Quality Broker allows for querying and improving data quality values. The Quality Notification Service is specifically targeted to the dissemination of changes on data quality values.

Key words: cooperative information system, data quality, XML model, data integration, publish & subscribe

⋆ This work is supported by the Italian Ministry for Education, University and Research (MIUR), through the COFIN 2001 Project "DaQuinCIS - Methodologies and Tools for Data Quality inside Cooperative Information Systems" (`http://www.dis.uniroma1.it/~dq/`). The DaQuinCIS project is a joint research project carried out by DIS - Università di Roma "La Sapienza", DISCo - Università di Milano "Bicocca" and DEI - Politecnico di Milano.

* Please send correspondence to Monica Scannapieco, *Dipartimento di Informatica e Sistemistica*, *Università di Roma "La Sapienza"*, Via Salaria 113 (piano 2, stanza 231), 00198 Roma, Italy. Telephone: +39 0649918479, fax: +39 0685300849.

Email addresses: `monscan@dis.uniroma1.it` (Monica Scannapieco), `virgi@dis.uniroma1.it` (Antonino Virgillito), `marchet@dis.uniroma1.it` (Carlo Marchetti), `mecella@dis.uniroma1.it` (Massimo Mecella),

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 [1]. 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 [2], 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. 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 interested organizations.

In this paper, we propose an architecture for managing data quality in CIS's. The architecture aims to avoid dissemination of low qualified data through the CIS, by providing a support for data quality diffusion and improvement. To 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's, both modeling and architectural issues.

The TDQM_CIS methodological cycle [3] defines the methodological framework in which the proposed architecture fits. The TDQM_CIS cycle derives from the extension of the TDQM cycle [4] to the context of CIS's; it consists of the five phases of Definition, Measurement, Exchange, Analysis and Improvement (see Figure 1).

The *Definition* phase implies the definition of a model for the data exported by each cooperating organization and of the quality data associated to them. Quality is a multi-dimensional concept: *data quality dimensions* characterize properties that are inherent to data [5,6]. Both data and quality data can be exported as XML documents, as further detailed in Section 3. The *Measure-*

baldoni@dis.uniroma1.it (Roberto Baldoni).

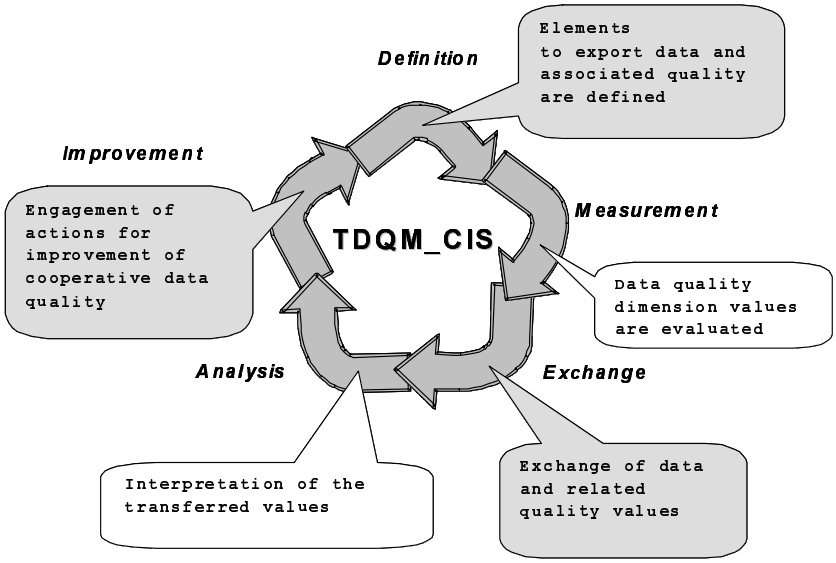


Fig. 1. The phases of the TDQM_CIS cycle; grey phases are specifically considered in this work

ment phase consists of the evaluation of the data quality dimensions for the exported data. The *Exchange* phase implies the exact definition of the exchanged information, consisting of data and of appropriate quality data corresponding to values of data quality dimensions. The *Analysis* phase regards the interpretation of the quality values contained in the exchanged information by the organization that receives it. Finally, the *Improvement* phase consists of all the actions that enable improvements of cooperative data by exploiting the opportunities offered by the cooperative environment. An example of improvement action is based on the analysis phase described above: interpreting the quality of exchanged information gives the opportunity of sending accurate feedbacks to data source organizations, which can then implement correction actions to improve their quality.

The architecture presented in this paper consists of different modules supporting each of the described phases. Nevertheless, in this paper we focus on two specific modules, namely the Data Quality Broker and the Quality Notification Service, which mainly support the Exchange and Improvement phases of the TDQM_CIS cycle. A model specifically addressing the *Definition* phase is also discussed. In Figure 1, the phases of the TDQM_CIS cycle specifically addressed in this paper are grey-colored.

The structure of the paper is as follows. In Section 2, a framework for Cooperative Information Systems is described, and the proposed architecture is outlined; furthermore, a running example for exemplifying the contributions is introduced; the example stems from the Italian *e*-Government scenario. In Section 3, the model for both the exchanging of data and their quality is presented. In Section 4 the Data Quality Broker is described and its distributed

design is discussed. In Section 5 the Quality Notification Service is illustrated. In 6, related research work is presented. Finally, Section 7 concludes the paper and points out the possible future extensions and improvement of the architecture.

2 A Cooperative Framework for Data Quality

In current government and business scenarios, organizations start cooperating 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 [2].

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* $\{ Org_1, \ldots, Org_n \}$ *that cooperate through a communication infrastructure* $\mathbb{N}$*, which provide software services to organizations as wells as reliable connectivity. Each organization* Org_i *is connected to* $\mathbb{N}$ *through a gateway* G_i*, on which software services offered by* Org_i *to other organizations are deployed. A* user *is a software or human entity residing within an organization and using the cooperative system.*

Several CIS's are characterized by a 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 administrations. But in such scenarios, the different organizations can provide the same data with different quality levels; thus any user of data may appreciate to exploit the data with the highest quality level, among the provided ones.

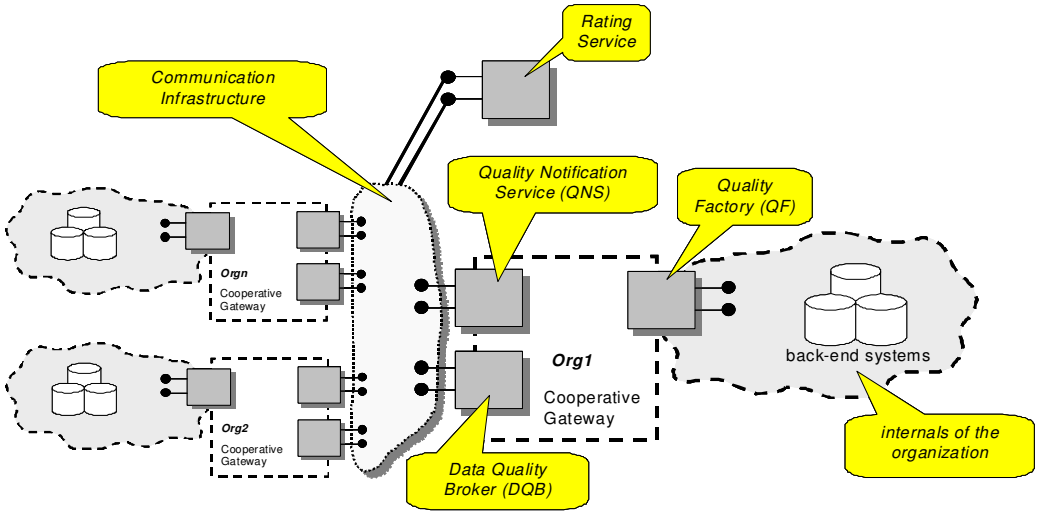


Fig. 2. The DaQuinCIS architecture

Therefore only the highest quality data should be returned to the user, limiting the dissemination of low quality data. Moreover, the comparison of the gathered data values might be used to enforce a general improvement of data quality in all organizations.

In the context of the DaQuinCIS project [1], we are proposing an architecture for the management of data quality in CIS's; this architecture allows the diffusion of data and related quality and exploits data replication to improve the overall quality of cooperative data. Each organization offers services to other organizations on its own cooperative gateway, and also specific services to its internal back-end systems. Therefore, cooperative gateways interface both internally and externally through services. Moreover, the communication infrastructure itself offers some specific services. Services 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 of the specific activities to be carried out. The overall architecture is depicted in Figure 2.

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. The D^2Q model is described in Section 3.

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

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

that is responsible for evaluating the quality of its own data. The design of the Quality Factory is out of the scope of this paper and has been addressed in [7].

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 function*. Different copies of the same data received as responses to the request are reconciled and a best-quality value is selected and proposed to organizations, that can choose to discard their data and to adopt higher quality ones; this is referred to as *quality improvement function*. If the requirements specified in the request cannot be satisfied, then the broker initiates a negotiation with the user that can optionally weaken the constraints on the desired data.

The Data Quality Broker is in essence a peer-to-peer data integration system [8] which allows to pose quality-enhanced query over a global schema and to select data satisfying such requirements. The Data Quality Broker is described in Section 4.

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. The Quality Notification Service is described in Section 5. Also the Quality Notification Service is deployed as a peer-to-peer system.

The **Rating Service** associates trust values to each data source in the CIS. These values are used to determine the reliability of the quality evaluation performed by organizations. The Rating Service is a centralized service, to be provided by a third-party organization. The design of the Rating Service is out of the scope of this paper; in this paper, for sake of simplicity, we assume that all organizations are reliable, i.e., provide trustable values with respect to quality of data.

2.1 Running Example

In this paper, we use a running example in order to show the proposed ideas. The example is a simplified scenario derived from the Italian Public Administration setting (interested reader can refer to [9] for a precise description of the real world scenario). Specifically, three agencies, namely the Social Security Agency, referred to as INPS (Istituto Nazionale Previdenza Sociale), the Ac-

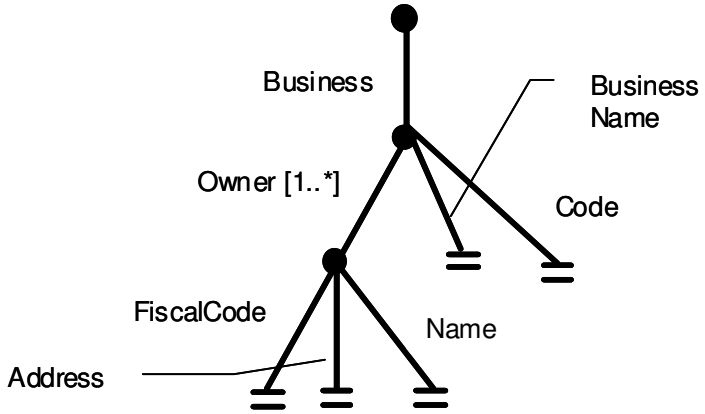
cident Insurance Agency, referred to as INAIL (Istituto Nazionale Assistenza Infortuni sul Lavoro), and the Chambers of Commerce, referred to as CoC (Camere di Commercio), together offer a suite of services designed to help businesses and enterprises to fulfill their obligations towards the Public Administration. Typically, businesses and enterprises must provide information to the three agencies when well-defined business events [2] occur during their lifetime, and they must be able to submit enquiries to the agencies regarding various administrative issues.

In order to provide such services, different data are managed by the three agencies: some data are agency-specific information about businesses (e.g., employees social insurance taxes, tax reports, balance sheets), whereas others are information that is common to all of them. 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.

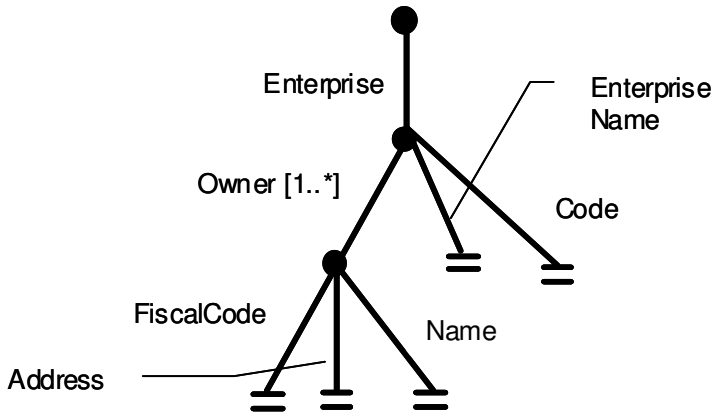
Each agency makes a different use of different pieces of the common information. As a consequence, each agency enforces different types of quality control that are deemed adequate for the local use of the information. Because each business reports the common data independently to each agency, the copies maintained by the three agencies may have different levels of quality.

In order to improve the quality of the data, and thus to improve the service level offered to businesses and enterprises, the agencies can deploy a cooperative information system according to the DaQuinCIS architecture; therefore, each agency exports its data according to a local schema. In Figure 2, three simplified versions of the local schemas are shown. The notation used in the figure represents conceptual schemas according to a hierarchical graph-based structure (the reasons of this choice will be clarified in Section 3). Local schemas are represented as edge-labelled graphs, with edge labels standing for conceptual elements names. Besides names, edge labels can also specify the cardinality of the concept identified by the name with respect to a father concept, i.e., a concept identified by an edge incoming in the upper level node of the edge under consideration. As an example, in Figure 2, the "1..*" label associated the `Owner` label means that a `Business` element can have 1 to N `Owner`) elements. When the cardinality is not specified, "1..1" is assumed

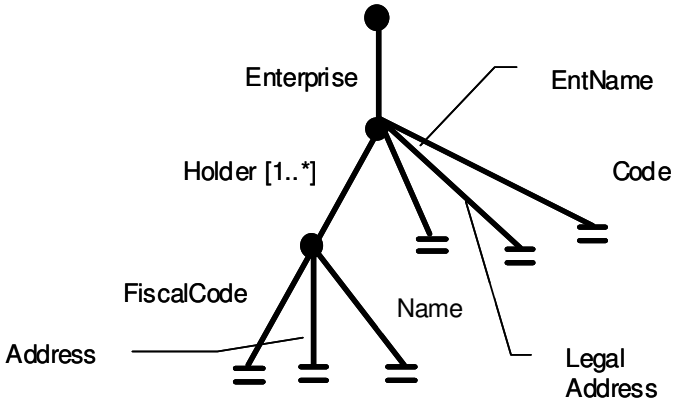
[2] Examples of business events are: the starting of a new business/company or the closing down, variations in legal status, board composition and senior management, variations in the number of employees, as well as opening a new location, filling for a patent, etc.



(a) Local schema of INAIL



(b) Local schema of INPS



(c) Local schema of CoC

Fig. 3. Local schemas used in the running example: conceptual representation

3 The D^2Q Model

According to 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*. In this section, we first introduce the data quality dimensions used in this paper (Section 3.1), then we describe the D^2Q model with respect to the data features (Section 3.2) and the quality features (Section 3.3).

3.1 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 [6].

In this section, we propose some data quality dimensions to be used in CIS's. The need for providing such definitions stems from the lack of a common reference set of dimensions in the data quality literature [5] and from the peculiarities of CIS's; we propose four dimensions, inspired by the already proposed definitions, and stemming from real requirements of CIS's scenarios that we experienced [2].

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: *(i)* accuracy, *(ii)* completeness, *(iii)* currency, and *(iv)* internal consistency.

3.1.1 Accuracy

In [6], accuracy refers to the proximity of a value v to a value v' considered as correct. More specifically:

Accuracy *is the distance between* v *and* v', *being* v' *the value considered as correct.*

Let us consider the following example: `Citizen` is a schema element with an attribute `Name`, and `p` is an instance of `Citizen`. If `p.Name` has a value $v =$ `JHN`, while $v' =$ `JOHN`, this is a case of low accuracy, as `JHN` is not an admissible value according to a dictionary of English names.

Accuracy can be checked by comparing data values with reference dictionaries (e.g., name dictionaries, address lists, domain related dictionaries such as product or commercial categories lists). As an example, in the case of values on string domain, edit distance functions can be used to support such a task [10]; such functions consider the minimum number of operations on individual characters needed in order to transform a string into another.

3.1.2 Completeness

Completeness *is the degree to which values of a schema element are present in the schema element instance.*

According to such a definition, we can consider: *(i)* the completeness of an attribute value, as dependent from the fact that the attribute is present or not; *(ii)* the completeness of a schema element instance, as dependent from the number of the attribute values that are present.

A recent work [11] distinguishes other kinds of completeness, namely: schema completeness, column completeness and population completeness. The measures of such completeness types can give very useful information for a "general" assessment of the data completeness of an organization, but, as it will be clarified in Section 3.3, our focus in this paper is on associating a quality evaluation on "each" data a cooperating organization makes available to the others.

In evaluating completeness, it is important to consider the meaning of null values of an attribute, depending on the attribute being mandatory, optional, or inapplicable: a null value for a mandatory attribute is associated with a lower completeness, whereas completeness is not affected by optional or inapplicable null values.

As an example, consider the attribute `E-mail` of the `Citizen` schema element; a null value for `E-mail` may have different meanings, that is *(i)* the specific citizen has no e-mail address, and therefore the attribute is inapplicable (this case has no impact on completeness), or *(ii)* the specific citizen has an e-mail address which has not been stored (in this case completeness is low).

3.1.3 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. Currency can be defined as:

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.*

More complex definitions of currency have been proposed in the literature [12]; they are especially suited for specific types of data, i.e, they take into account elements such as "volatility" of data (e.g., data such as stock price quotes that change very frequently). However, in this work, we stay with the provided general definition of currency, leaving its extensions to future work.

Currency can be measured either by associating to each value an "updating time-stamp" [13] or a "transaction time" like in temporal databases [14].

3.1.4 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:

Internal Consistency *is 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.*

As an example, if we consider `Citizen` with attributes `Name`, `DateOfBirth`, `Sex` and `DateOfDeath`, some possible semantic rules to be checked are:

- the values of `Name` and `Sex` for an instance `p` are consistent. If `p.Name` has a value v = `JOHN` and the value of `p.Sex` is `FEMALE`, this is a case of low internal consistency;
- the value of `p.DateOfBirth` must precede the value of `p.DateOfDeath`.

Internal consistency has been widely investigated both in the database area through integrity constraints (e.g., [15]) and in the statistics area, through edits checking (e.g., [16]).

3.2 Data Model

The D^2Q model adopts a database view of XML: an XML Document is a set of data items, and an Document Type Definition (DTD) is the schema of such

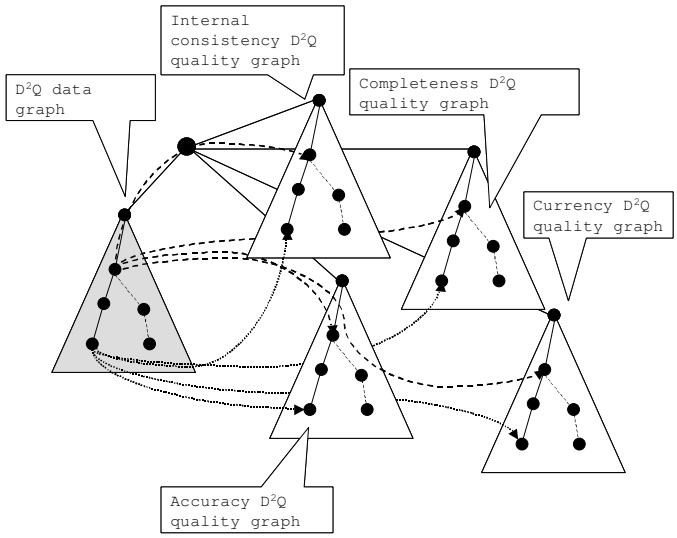


Fig. 4. The generic structure of a D^2Q XML document

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 3.1. 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 4. Organizations in the CIS export local schemas as D^2Q *XML DTD's*.

A D^2Q XML document corresponds to a set of conceptual data items, which are instances of conceptual schema elements, specified in the D^2Q XML DTD; 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 3.3.

As an example, consider the document `enterprises.xml`, shown in Figure 5, which contains entries about businesses with the associated quality data. Such a 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. Specifically, an instance of `Enterprise` and the related `Accuracy` values are depicted.

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

- *a name* δ *which is unique;*
- *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 *(which is unique) and* $type_i$ *can be:*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Enterprises SYSTEM "...\enterprises.dtd">
<Enterprises>
    <Enterprise Accuracy="o00" ...>
        <Code Accuracy="o01" ...>APTA</Code>
        <EntName Accuracy="o02" ...>Associazione Promozione Tecnologie Avanzate</EntName>
        <LegalAddress Accuracy="o03" ...>Borgo Vittorio 5, 00193 Roma, Italy</LegalAddress>
        <Holder Accuracy="o04" ...>
            <Name Accuracy="o05" ...>Massimo Mecella</Name>
            <Address Accuracy="o06" ...>Via dei Gracchi 71, 00192 Roma, Italy</Address>
            <FiscalCode Accuracy="o07" ...>MCLMSM73H17H501F</FiscalCode>
        </Holder>
        <Holder Accuracy="o08" ...>
            <Name Accuracy="o09" ...>Monica Scannapieco</Name>
            <Address Accuracy="o10" ...>Via Mazzini 27, 84016 Pagani (SA), Italy</Address>
            <FiscalCode Accuracy="o11" ...>SCNMNC74S70G230T</FiscalCode>
        </Holder>
    </Enterprise>
    <Accuracy_Enterprise OID="o00">
        <Accuracy_Code OID="o01">high</Accuracy_Code>
        <Accuracy_EntName OID="o02">high</Accuracy_EntName>
        <Accuracy_LegalAddress OID="o03">medium</Accuracy_LegalAddress>
        <Accuracy_Holder OID="o04">
            <Accuracy_Name OID="o05">high</Accuracy_Name>
            <Accuracy_Address OID="o06">high</Accuracy_Address>
            <Accuracy_FiscalCode OID="o07">high</Accuracy_FiscalCode>
        </Accuracy_Holder>
        <Accuracy_Holder OID="o08">
            <Accuracy_Name OID="o09">medium</Accuracy_Name>
            <Accuracy_Address OID="o10">medium</Accuracy_Address>
            <Accuracy_FiscalCode OID="o11">high</Accuracy_FiscalCode>
        </Accuracy_Holder>
    </Accuracy_Enterprise>
    <Completeness_Enterprise ...>
    ...
    </Completeness_Enterprise>
    <Currency_Enterprise ...>
    ...
    </Currency_Enterprise>
    <InternalConsistency_Enterprise ...>
    ...
    </ InternalConsistency_Enterprise>
</Enterprises>
```

Fig. 5. An example of D^2Q XML document.

- *either a basic type*[3] *;*
- *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 di-*

[3] 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`.

mension. 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*[4] *;*

- *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 $name_i$.*

As data class names and properties are unique, indeed a D^2Q data graphs is acyclic. In Figure 6, the D^2Q data graph of the running example is shown: an object instance `APTA` of the data class `Enterprise` is considered. The data class has `Code` and `EntName` as properties of basic types and a property of type `set-of < Holder >`; the data class `Holder` has all properties of basic types.

[4] The use of links will be further explained in Section 3.3, when quality graphs are introduced.

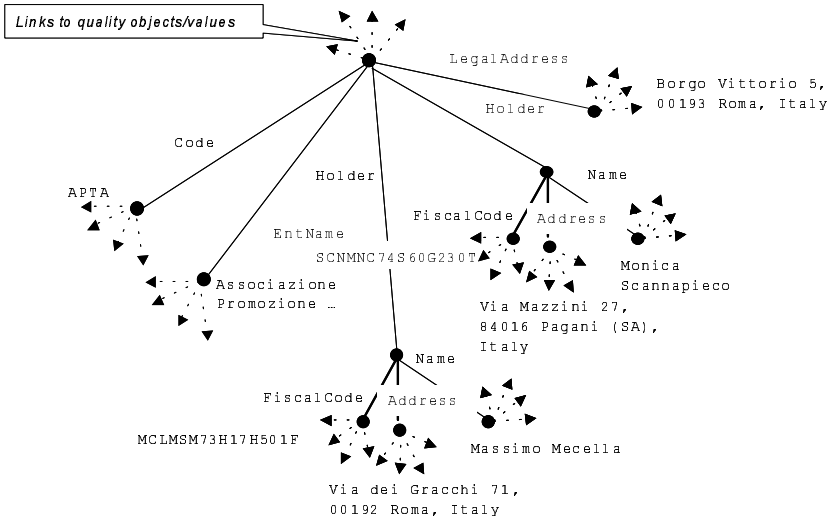


Fig. 6. The D^2Q data graph of the running example

3.3 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 3.1.

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., 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` $\}$*, which is unique;*
- *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*;*

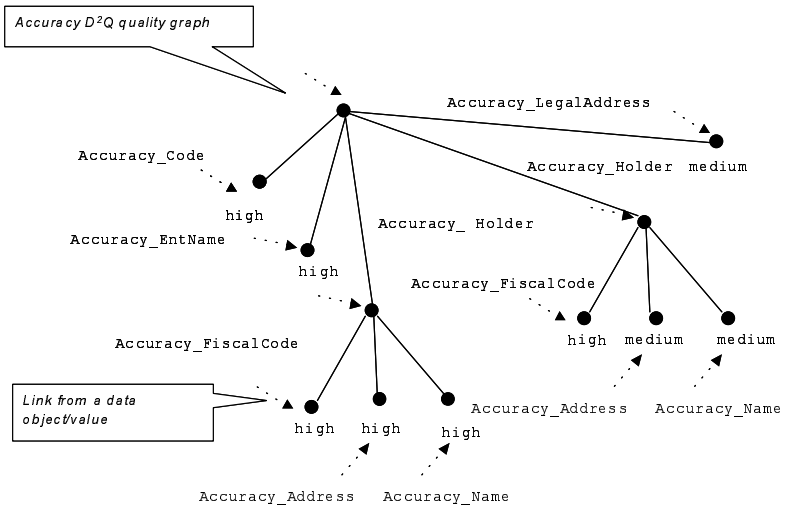


Fig. 7. The accuracy D^2Q quality graph of the running example

- *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 7, the D^2Q quality graph concerning accuracy of the running example is shown, and links are highlighted; for instance, the accuracy of `APTA` (i.e, which is the value of the `Code` element) is `0.9`.

Data and quality data are exchanged as D^2Q XML documents in the CIS. 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 3.1.

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 $\mathcal{G}_i$, $i = 1 \dots m$, each of them representing the data objects $\mathcal{O}_i$;*
- *$4 * m$ D^2Q quality graph $\mathcal{G}_i^D$, $i = 1 \dots m$, 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 with respect to other possible solutions, 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.

4 The Data Quality Broker

The *Data Quality Broker (DQB)* is the core component of the architecture. It is implemented as a peer-to-peer distributed service: each organization hosts a copy of the Data Quality Broker that interacts with other copies. We first provide an overview of the high-level behavior of the Data Quality Broker in Sections 4.1 and 4.2; then, in Sections 4.3 and 4.4, we explain the details of its design and implementation.

4.1 Overview

The general task of the Data Quality Broker is to allow users to select data in the CIS according to their quality. Users can issue queries on a D^2Q *global schema*, specifying constraints on quality values. Then, the Data Quality Broker selects, among all the copies of the requested data that are in the CIS, only those satisfying all specified constraints.

Quality constraints can be related to quality of *data* and/or quality of *service* (QoS). Specifically, besides the four data quality dimensions previously introduced, two QoS dimensions are defined, namely: *(i)* a *responsiveness*, defined by the average round-trip delay evaluated by each organization periodically pinging gateway in the CIS; and *(ii) availability* of a source, evaluated by

pinging a gateway and considering if a timeout expires; in such a case the source is considered not available for being currently queried.

A user can request data with a specified quality (e.g., "return enterprises with accuracy greater medium") or declare a limit on the query processing time, restricting the query only to available sources (e.g., "return only enterprises from the first responding source") or include both kinds of constraints (e.g., "return enterprises with accuracy greater than medium provided by the first responding source").

The Data Quality Broker performs two specific tasks:

- *query processing* and
- *quality improvement.*

The semantics of query processing is described in Section 4.2, while how query processing is performed is described in Section 4.3.2. The quality improvement feature consists of notifying organizations with low quality data about higher quality data that are available through the CIS. This step is enacted each time copies of the same data with different quality are collected during the query processing activity. The best quality copy is sent to all organizations having lower quality copies. Organizations involved in the query have the choice of updating or not their data. This gives a non-invasive feedback that allows to enhance the overall quality of data in the CIS, while preserving organizations' autonomy. The details of how the improvement feature is realized are provided in Section 4.3.

4.2 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 [17,8]. Both the global schema and local schemas exported by cooperating organizations are expressed according to the D^2Q model. The adopted query language is XQuery [18]. The unfolding of an XQuery query issued on the global schema can be performed on the basis of well-defined mappings with local sources. Details concerning such a mapping are beyond the scope of this paper (details can be found in [19]).

Under our hypothesis, data sources have distinct copies of the same data having 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.

Specifically, the detailed steps we follow to answer a query with quality constraints are:

(1) let $\mathcal{Q}$ be a query posed on the the global schema $\mathbb{G}$;
(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 $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$ queries to be posed over local sources;
(3) The execution of the queries $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$ returns a set of results $\mathcal{R}_1, \ldots, \mathcal{R}_n$. On such a set an *extensional correspondence* property is checked, namely: given $\mathcal{R}_i$ and $\mathcal{R}_j$, the items common to both results are identified, by considering the items referring to the same objects. This step is performed by using a well-known algorithm for record matching [20].
(4) The result to be returned is built as follows: *(i)* if no quality constraint is specified, a best quality default semantics is adopted. This means that the result is constructed by selecting the best quality values (this step is detailed in 4.3.2); *(ii)* if quality constraints are specified, the result is constructed by checking the satisfiability of the constraints on the whole result. Notice that the quality selection is performed at a property level, on the basis of best (satisfying) quality values.

4.2.1 Example of Query Processing

In the scenario proposed in Section 2.1, let us assume that the global schema derived from the INPS, INAIL and CoC local schemas is the one proposed in Figures 8 and 9 [5]. Some examples of mapping info between global and local schemas are shown in Table 5.

Let us consider the following XQuery query to be posed on the global schema.

```
for $b in document("GlobalSchema.xml")//Enterprise
where $b/Code ="APTA"
return <Holder> { $b/Name } { $b/Address }</Holder>
```

On the basis of mapping information, the unfolding of concepts in the global schema is performed by substituting the semantically corresponding concept in

[5] It is worth noting that the XML documents exported locally are conform to the D^2Q model, hence they have a well-defined structure consisting of graphs linking classes and literals. Moreover, the reader should notice that the XML document presented in Figure 5 is valid according to such a global schema.

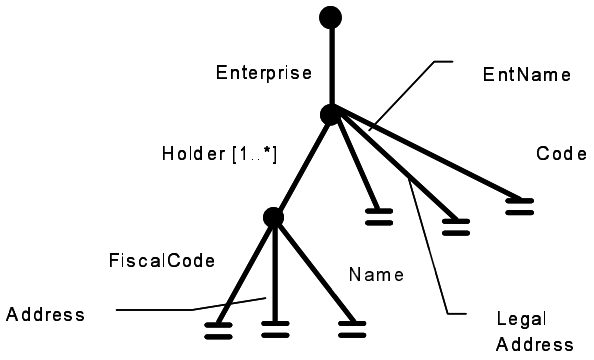


Fig. 8. Graphical representation of the global schema

Table 1
Mapping between global and local schemas

Global schema	**INPS schema**	**INAIL schema**	**CoC schema**
Business	Business	Enterprise	Business
Business/ID	Business/ID	Enterprise/ID	Business/ID
Holder	Owner	Holder	Owner
Holder/Name	Owner/Name	Holder/Name	Owner/Name
...	...	...	...
LegalAddress	X	X	LegalAddress
...	...	...	...

the local schema. The schema mapping information shown in 5 is a simplified version of the actual mapping information that includes the specification of path expressions exactly locating concepts in the local schemas. Therefore the unfolding process requires some specific technicalities; we have implemented the complete unfolding and refolding process, details of which can be found in [19]. As an example, the following local query is posed on INAIL:

```
for $b in document("Inail.xml")//Enterprise
where $b/Code ="APTA"
return <Holder> { $b/Name } { $b/Address }</Holder>
```

Results produced by executing XQuery queries locally to INPS, INAIL and CoC include both data and associated quality. Table 5 illustrates such results. Each organization returns two holders for the specified business, with the associated accuracy values. As an example, accuracy of properties `Name` and `Address` of the first `Holder` item provided by INAIL are respectively `low` and `medium`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Enterprises (Enterprise*, Accuracy_Enterprise*, ...)>
<!ELEMENT Enterprise (Code, EntName, LegalAddress, Holder+)>
<!ATTLIST Enterprise
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT Code (#PCDATA)>
<!ATTLIST Code
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT EntName (#PCDATA)>
<!ATTLIST EntName
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT LegalAddress (#PCDATA)>
<!ATTLIST LegalAddress
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT Holder (Name, Address, FiscalCode)>
<!ATTLIST Holder
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT Name (#PCDATA)>
<!ATTLIST Name
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT Address (#PCDATA)>
<!ATTLIST Address
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT FiscalCode (#PCDATA)>
<!ATTLIST FiscalCode
    Accuracy IDREF #IMPLIED
    ...
>
<!ELEMENT Accuracy_Enterprise (Accuracy_Code, Accuracy_EntName, Accuracy_LegalAddress, Accuracy_Holder+)>
<!ATTLIST Accuracy_Enterprise
    OID ID #REQUIRED
>
<!ELEMENT Accuracy_Code (#PCDATA)>
<!ATTLIST Accuracy_Code
    OID ID #REQUIRED
>
<!ELEMENT Accuracy_EntName (#PCDATA)>
<!ATTLIST Accuracy_EntName
    OID ID #REQUIRED
>
<!ELEMENT Accuracy_LegalAddress (#PCDATA)>
<!ATTLIST Accuracy_LegalAddress
    OID ID #REQUIRED
>
<!ELEMENT Accuracy_Holder (Accuracy_Name, Accuracy_Address, Accuracy_FiscalCode)>
<!ATTLIST Accuracy_Holder
    OID ID #REQUIRED
>
<!ELEMENT Accuracy_Name (#PCDATA)>
<!ATTLIST Accuracy_Name
    OID ID #REQUIRED
>
<!ELEMENT Accuracy_Address (#PCDATA)>
<!ATTLIST Accuracy_Address
    OID ID #REQUIRED
>
<!ELEMENT Accuracy_FiscalCode (#PCDATA)>
<!ATTLIST Accuracy_FiscalCode
    OID ID #REQUIRED
>
...
```


Fig. 9. The global DTD

Table 2
Query results in a sample scenario

Organization	`Holder` **item 1**	**Accuracy item 1**	`Holder` **item 2**	**Accuracy item 2**
INAIL	"Maimo Mecella"	Acc_Name = low	"Monica Scanapieco"	Acc_Name = medium
	"Via dei Gracchi 71, 00192 Roma, Italy"	Acc_Address = high	"Via Mazzini 27, 84016 Pagani"	Acc_Address = medium
INAIL	"M. Mecalla"	Acc_Name = low	"Monica Scannapieco"	Acc_Name = high
	"Via dei Gracchi 71, Roma, Italy"	Acc_Address = medium	"Via Mazzini 27, 84016 Pagani (SA), Italy"	Acc_Address = high
CoC	"Massimo Mecella"	Acc_Name = high	"M. Scanna"	Acc_Name = low
	"Via dei Gracchi 71, 00192 Roma, Italy"	Acc_Address = high	"Via Mazzini 27, Italy"	Acc_Address = low

Once results are gathered from INPS, INAIL and CoC, a record matching process is performed. In [20], an algorithm for record matching was proposed for large data bases, based on the idea of comparing only records included in a sliding window, in order to establish their matching. Such a method is known as Sorted Neighborhood Method (SNM), and consists of three distinct steps: *(i)* choice of the matching key; *(ii)* sorting of records according to the chosen key, thus originating clusters of records that have the same key value as aggregating element; *(iii)* moving of a fixed size window through the list of records in each cluster and comparisons only of the records included in the window. We adopt this algorithm in order to carry on the record matching process.

Specifically, data objects returned by an organization are tentatively mapped with data objects returned by the other ones. In the example, by using `Name` as a key for matching, it is easy to find two clusters, namely the `Holder` item 1 cluster and the `Holder` item 2 cluster. Inside each cluster, an instance level reconciliation is performed on the basis of quality values.

According to the shown accuracy values, the result is composed as follows:

`Holder.Name` = ‘‘`Massimo Mecella`’’

from CoC as its accuracy = `max(`*Acc_Name(INAIL)*, *Acc_Name(INPS)*, *Acc_Name(CoC)*`)` = *Acc_Name(CoC)* = `high`.

and

`Holder.Address` = ‘‘`Via dei Gracchi 71, 00192 Roma, Italy`’’

from INPS as its accuracy = `max(`*Acc_Address(INAIL)*, *Acc_Address(INPS)*, *Acc_Address(CoC)*`)` = *Acc_Address(INPS)* = `high`.

Similar considerations lead to the selection of the `Holder` item 2 provided by INPS.

4.3 Internal Architecture and Deployment

In this section we first detail the design of the Data Quality Broker, by considering its interaction modes and its constituting modules; then, an example of query processing execution is shown.

4.3.1 Interaction Modes and Application Interface

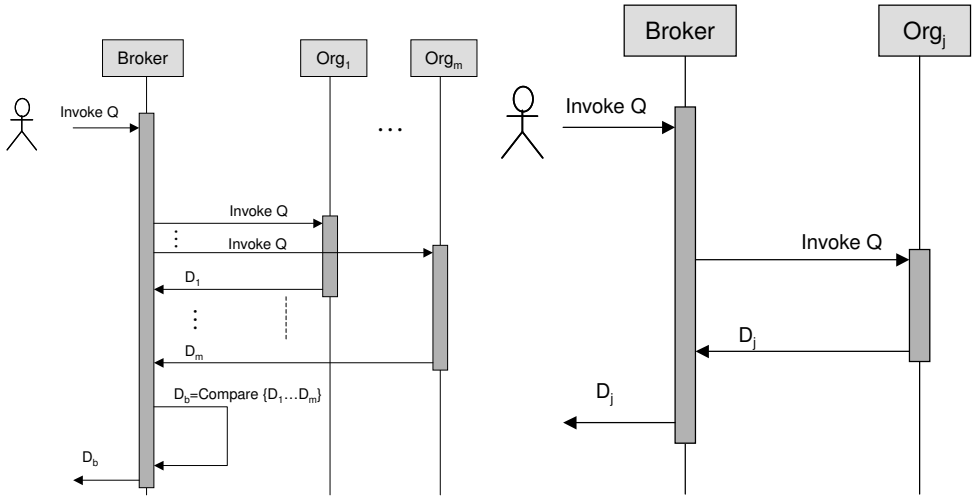
The Data Quality Broker exports the following interface to its clients:

- `invoke(query, mode)`, for submitting a query $\mathcal{Q}$ over the global schema including quality constraints. The behavior of the broker can be set with the `mode` parameter; such a parameter can assume either the `PESSIMISTIC` or `OPTIMISTIC` value.
  In the *pessimistic* mode, the broker accesses all sources that provide data satisfying the query and builds the result, whereas in the *optimistic* mode the broker accesses only those organizations that *probably* will provide good quality data. Such information can be derived on the basis of statistics on quality values. Note that in the optimistic mode, the broker does not contact all sources and does not have to wait for all responses; however, this optimization can lead to non accurate results. Conversely, in the pessimistic mode, all sources are always contacted; it takes more time to retrieve the final result, but it is always the most accurate one.
  Figure 10 shows the sequence of operations performed in each of the two modes: in pessimistic mode (Figure 10(a)), all organizations storing data satisfying a query are queried and all results have to be waited for before proceeding. After results are retrieved, a comparison phase is required to choose the best data. In optimistic mode (Figure 10(b)) only the organization Org_j that has the best overall statistical quality evaluation is involved in the query and no comparison is required.
- `propose(data)`, for notifying all sources that have previously sent data with lower quality than the selected one, with higher quality results. This function can be only invoked in the pessimistic mode. Each time a query is posed, different copies of same data are received as answers; the broker submits to organizations the best quality copy $\mathcal{R}$ selected among the received ones. The `propose()` operation is invoked by a peer instance of the broker in order to provide the quality feedback.

4.3.2 Modules of the Data Quality Broker

The Data Quality Broker is internally composed of five interacting modules (Figure 11). The Data Quality Broker is deployed as a peer-to-peer service: each organization hosts an independent copy of the broker and the overall service is realized through an interaction among the various copies, performed via the Transport Engine module. The modules Query Engine, Transport Engine and Schema Mapper 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 Comparator can also be customized to implement different criteria for quality comparison.

Query Engine (QE): receives a query from the client and processes it. The query is executed issuing sub-queries to local data sources. The QE can also perform some query optimizations during the processing.



(a) Pessimistic (b) Optimistic

Fig. 10. Broker invocation modes

Wrapper (Wr): 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, that is it returns data stored inside organizations without modifying them.

Transport Engine (TE): communication facility that transfers queries and their results between the QE and data source wrappers. It also evaluates the QoS parameters and provides them to the QE for being used in query optimization.

Schema Mapper (SM): stores the static mappings between the global schema and the local schemas of the various data sources.

Comparator (CMP): compares the quality values returned by the different sources to choose the best one.

In the following, we detail the behaviour of QE and TE. These are the modules that required most design effort, due to their higher complexity. The SM stores mappings between the global and local schemas, the Wr is a classical database wrapper, and the CMP compares quality value vectors according to ranking methods known in literature, such as Simple Additive Weighting (SAW) [21] or Analytical Hierarchy Process (AHP) [22].

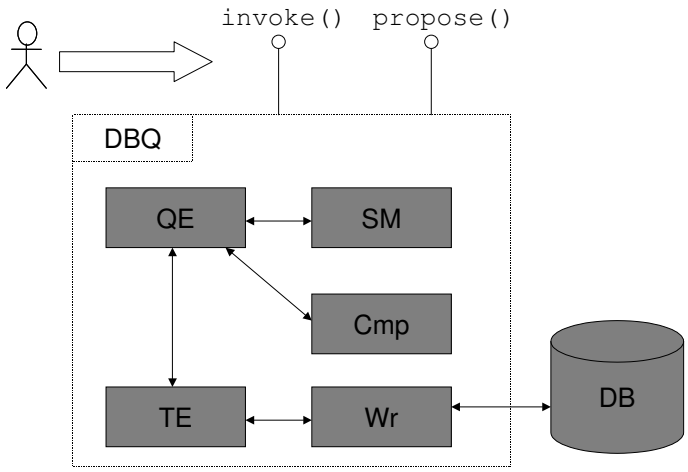


Fig. 11. Broker modules

The Query Engine is responsible for query execution and optimization. The copy of the Query Engine local with the user, receives the query and splits the query in sub-queries (local to the sources) by interacting with the Schema Mapper (which manages the mapping between the global schema and the local schemas). Then, the local QE also interacts with the TE in order to: *(i)* send local queries to other copies of the QE and receive the answers; *(ii)* be notified about QoS parameters to be used for optimization.

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

- Query execution (request/reply interaction): the requesting TE sends a query to the local TE of the target data source and remains blocked waiting for the result. A request is always synchronous but it can be always stopped from the outside, for example when the QE decides to block the execution of a query to enforce QoS constraints. In this case, the results are discarded.
- Quality feedback (asynchronous interaction): when a requesting QE has selected the best quality result of a query, it contacts the local TE's to enact quality feedback propagation. The `propose()` operation 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 TE simply invokes this operation, whose behaviour depends from the strategy that an organization chooses for the improvement of its own data on the basis of external notifications.

The other function performed by the TE is the evaluation of the QoS parameters. This feature is encapsulated into the TE as it can be easily implemented exploiting TE's communication capabilities. Specifically, the TE periodically pings data sources in order to calculate responsiveness. Moreover, the timeout value necessary to calculate the availability QoS parameter is configured in

the TE.

4.3.3 Query Processing in the Pessimistic Mode

In this section we give details of pessimistic mode query invocations, in order to clarify the behavior of the Data Quality Broker during a query. Specifically, we describe the interaction among peer Data Quality Brokers and among modules inside each Data Quality Broker when a query with quality requirements is submitted by a user.

After receiving the query, the Query Engine (local to the client) interacts with the Schema Mapper and generates a set of sub-queries, which in turn are sent through the local Transport Engine to all Transport Engines of the involved organizations. All Transport Engines transfer the query to local Query Engines that execute local queries by interacting with local Wrappers. Query results are passed back to the local Transport Engine that passes all results to the local Query Engine. This module selects the best quality result through the Comparator and passes it to the client. Once the best quality data is selected the feedback process starts.

A second phase involves all sources that have previously sent data with lower quality than the selected one; each source may adopt the proposed data (updating its local data store) or not. Indeed criteria to decide whether updating the local data store with the feedback data may vary from one organization to another; such criteria can be set by providing proper implementations of the `propose()` operation.

Figure 12 depicts an invocation by a client of the organization Org_1, specifically the `invoke(''Select APTA from Enterprises with accuracy` $\geq$ `medium'', PESSIMISTIC)`. For the sake of clarity, only involved modules are depicted. The sequence of actions performed during the query invocation is described in the following. We indicate with a subscript the specific instance of a module belonging to a particular organization: for example, QE_i indicates the Query Engine of organization Org_i.

(1) QE_1 receives the query $\mathcal{Q}$ posed on the global schema, as a parameter of the `invoke()` function;
(2) QE_1 retrieves the mapping with local sources from SM_1;
(3) QE_1 performs the unfolding that returns queries for Org_2 and Org_3 that are passed to TE_1;
(4) TE_1 sends the request to Org_2 and Org_3;
(5) The recipient TE's pass the request to the Wr's modules. Each Wr accesses the local data store to retrieve data;
(6) Wr_3 retrieves a query result, e.g., $\mathcal{R}_a$ *with accuracy 0.7*, and Wr_2 retrieves another query result $\mathcal{R}_b$ *with accuracy 0.9*;

(7) Each TE sends the result to TE_1;
(8) TE_1 sends the collected results ($\mathcal{R}_a$, $\mathcal{R}_b$) to QE_1. QE_1 selects through CMP_1 the result with the greatest accuracy, i.e., Org_2's result ($\mathcal{R}_b$ *with accuracy 0.9*);
(9) QE_1 sends the selected result to the client;
(10) QE_1 starts the feedback process sending the selected result ($\mathcal{R}_b$ *with accuracy 0.9*) to TE_1;
(11) TE_1 sends it to Org_3;
(12) TE_3 receives the feedback data and makes a call `propose(`$\mathcal{R}_b$ *with accuracy 0.9*`)`.

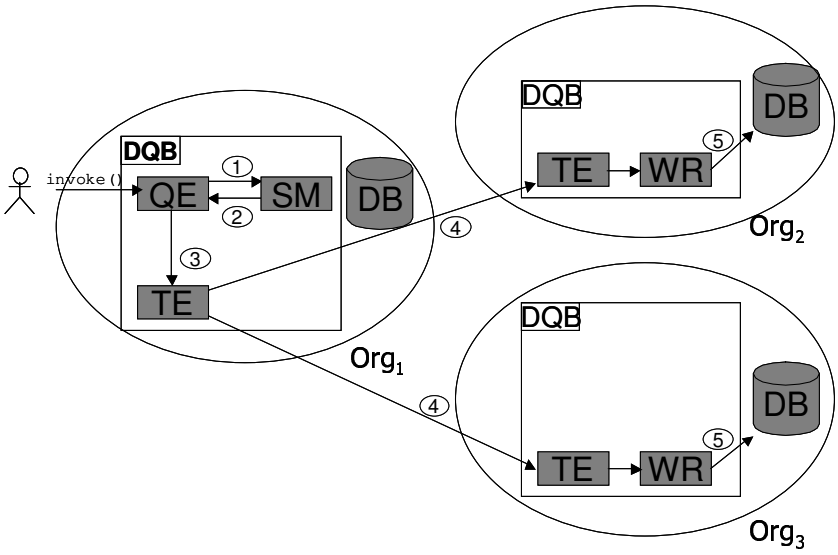
4.4 Implementation

We have developed a prototype of the broker, implementing the pessimistic query invocation mode. In our prototype, the Data Quality Broker is realized as a Web Service [23], deployed on all the organizations. A Web Service-based solution is suitable for integrating heterogeneous organizations as it allows to deploy the Data Quality Broker on the Internet, regardless from the access limitations imposed by firewalls, interoperability issues, etc. Moreover, by exploiting standard Web Service technologies, the technological integration among different organizations is straightforward; specifically, we tested interoperability among different versions of the Data Quality Broker, realized using respectively the JAX-RPC [6] framework and the .NET one [7].

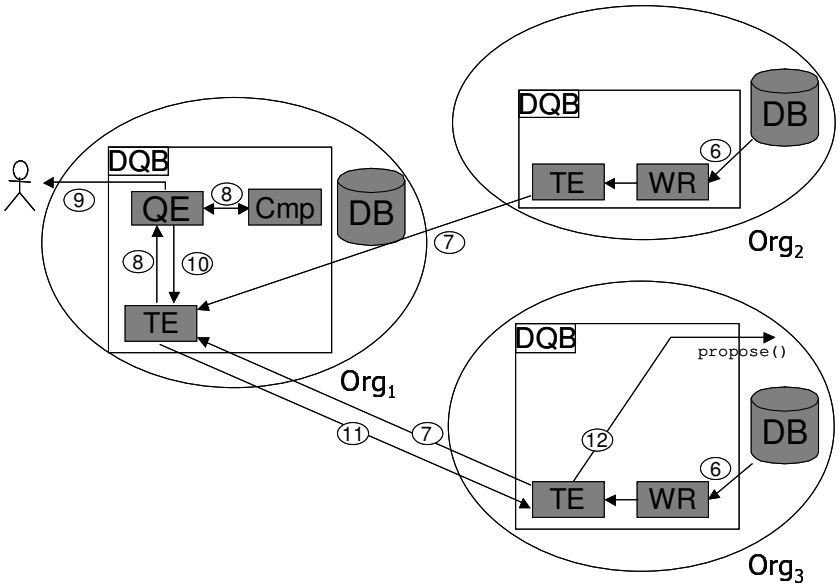
As we discussed above, the component responsible for the communication between the Data Quality Broker instances is the Transport Engine. Besides the operations previously described (i.e., `propose()` and `invoke()`), it exposes also the `ping()` operation, which is used to estimate the query responsiveness. Such operations are invoked through standard SOAP messages sent over HTTP; specifically, invocations to multiple Transport Engines are performed in parallel. Remote invocations to the `invoke()` operation require the client Transport Engine to wait for all results.

If we assume that Data Quality Broker instances are subject to faults this can cause the client to wait indefinitely for responses from faulty instances. From the distributed system theory stems that in communication systems, such as the Internet, where delays on message transfer cannot be predicted, it is impossible to distinguish a faulty software component from a very slow one [24,25]. Then, the client Transport Engine sets a timeout for each invoked source. The timeout value is estimated for each single source on the basis of its responsiveness. After the timeout expires, no result is returned for that

[6] `http://java.sun.com/xml/jaxrpc/`
[7] `http://msdn.microsoft.com/netframework/`



(a) Step 1 – 5



(b) Step 6 – 12

Fig. 12. Details of query invocation in the pessimistic mode

query. This is a best-effort semantics that does not guarantee to always return the best quality value but rather enforces the correct termination of each query invocation. Replication techniques can be exploited in order to enhance the availability of each web service, ensuring more accurate results even in presence of faults [**?**].

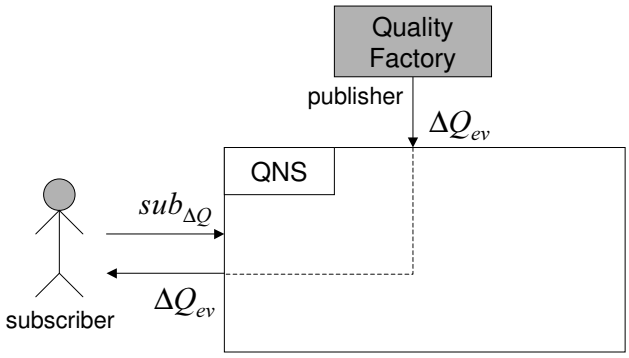


Fig. 13. Quality Notification Service

5 The Quality Notification Service

The Quality Notification Service (QNS) is used to inform interested users when changes in quality values occur within the CIS.

The interaction between the Quality Notification Service and a user follows the publish/subscribe paradigm: a user willing to be notified for quality changes *subscribes* to the Quality Notification Service by submitting the features of the events to be notified for, through a specific subscription language. When a change in quality happens, an event is *published* by the Quality Notification Service i.e., all the users which have a consistent subscription receive a notification.

The Quality Notification Service can be in the CIS to control the quality of critical data, e.g., in order to keep track of its quality changes and be always aware when quality degrades under a certain threshold, which makes it no longer suited for the use it is devoted to. The Quality Notification Service can also exploited by other architectural components to maintain updated the information they use to perform their services.

In this section we first introduce the QNS specification, i.e., *(i)* the language accepted by the Quality Notification Service to let user specify the set of notifications they are interested in, and *(ii)* the form of quality change notifications received by subscribed users; then we motivate and detail the internal architecture of the service. The explanation makes use of the running example introduced in Section 2.1.

5.1 Specification

Figure 13 illustrates a high-level depiction of the Quality Notification Service, and its relationships with users and with the Quality Factory. A *quality change*

occurs within an organization each time the value of a given quality dimension of a property varies [8]. Quality changes are managed by the Quality Factory of each specific organization in a way that depends on the internal policies and infrastructures of the organization. Upon each quality change the Quality Factory reports to the Quality Notification Service the data object involved in the change (according to the data representation of the global schema) along with the associated quality data. As aforementioned, the Quality Notification Service is thus in charge of notifying the new data to all interested users, according to the values of the quality data.

Subscription language. The Quality Notification Service subscription language has to allow quality-based subscriptions where quality dimensions are constrained by threshold values. In order to provide users with a flexible subscription language, the subscription granularity should allow to express interest in quality changes of sets of data objects. As examples, an user could be interested in monitoring accuracy changes of enterprises located in Rome whose records are stored throughout the whole CIS, whether another user could will to subscribe for currency changes of a specific enterprise whose records are stored in a specific organization.

Therefore, a generic Quality Notification Service subscription is a triple in the form:

$$sub_{\Delta Q} = \langle \delta[: \mathcal{S}], expr_D, expr_Q \rangle$$

where:

- δ is a data class expressed on the global schema;
- $\mathcal{S}$ is an optional set of sources for the data class δ, i.e., organizations containing data objects belonging to δ according to the mapping rules;
- $expr_D$ is an expression used to specify the set of data objects of class δ whose quality changes are relevant to the user. It is a conjunction of constraints on some properties of δ. A constraint is a triple $\langle p, op, v \rangle$ where p is a property of δ, op is an operator (e.g. $=, <, >, ...$) depending on the type of p, and v is a (possibly empty) value of the same type of p;
- $expr_Q$ is an expression used to select notifications for quality changes occurred on data objects selected by $\langle \delta[: \mathcal{S}], expr_d \rangle$ according to the values of the quality data. Even this expression is a conjunction of constraints, i.e., a triple $\langle p.qd, op, t \rangle$ where $p.qd$ is a quality dimension of property p of data class δ, op is a comparison operator ($=, <, >, \leq, \geq$), and v is a numeric value ranging from 0 to 1;

[8] Let us remark that each attribute has an associated set of quality dimensions.

At least δ must be specified, while other parameters can be left empty (i.e., set to $\perp$) [9].

Once a user has subscribed for some notifications, he will receive them in the same form they are produced by quality factory within each organization, that is described below.

Notifications of quality changes. The Quality Factory component of each organization pushes events to the Quality Notification Service in the following form:

$$\Delta_{Q_{ev}} = \langle \delta,\ d,\ \{\ldots \langle qd,\ v\rangle_i \ldots\}\ \rangle$$

where δ is a data class of the global schema, d is a data object of class δ, and the last element is a set of pairs $\langle qd,\ v\rangle$, where qd is a quality dimension and v is the value of the associated quality data, that express the new values of the quality attributes of data object d.

Subscription Matching and Containment. As aforementioned, users will receive only those notifications that *match* their subscriptions. A notification $N = \langle \delta_N,\ d_N,\ \{\ldots \langle qd,\ v\rangle_i \ldots\}\ \rangle$ produced by a source s *matches* a subscription $S = \langle \delta_S[: \mathcal{S}], expr_D, expr_Q\rangle$ *iff*:

(1) the data classes of N and S coincide, i.e. $\delta_N = \delta_S$, and if $\mathcal{S}$ is specified (i.e., $\mathcal{S} \neq \perp$) then $s \in \mathcal{S}$;
(2) the data object d_N satisfies $expr_D$ of S;
(3) the set $\{\ldots \langle qd,\ v\rangle_i \ldots\}$ satisfies $expr_Q$ of S.

Notification matching against a set of subscriptions can be efficiently evaluated using well known algorithms, e.g., [26,27].

We now introduce the concept of *containment* between subscriptions. As shown in the following, subscription containment is useful in order to reduce both the use of memory and the network traffic due to the services provided by the Quality Notification Service. Intuitively, a subscription S_1 contains another subscription S_2 if all notifications that match S_2 also match S_1. More formally, given two subscriptions $S_1 = \langle \delta_1[: \mathcal{S}_1], expr_{D_1}, expr_{Q_1}\rangle$ and $S_2 = \langle \delta_2[: \mathcal{S}_2], expr_{D_2}, expr_{Q_2}\rangle$, then S_1 *contains* S_2 *iff*:

(1) δ_1 is an ancestor node of δ_2 in the D^2Q acyclic graph and if both $\mathcal{S}_1 \neq \perp$ and $\mathcal{S}_2 \neq \perp$ then $\mathcal{S}_2 \subseteq \mathcal{S}_1$;

[9] For the sake of simplicity, we omit to present aspects related to user unsubscriptions.

(2) for each data object d that satisfies $expr_{D_2}$, d also satisfies $expr_{D_1}$;
(3) analogously, for each set $\{\ldots\langle qd,\ v\rangle_i\ldots\}$ that satisfies $expr_{Q_2}$, also $expr_{Q_1}$ is satisfied by $\{\ldots\langle qd,\ v\rangle_i\ldots\}$.

It is possible to show that subscription containment is a transitive relationship among subscriptions, i.e. if S_1 contains S_2 that contains S_3 then S_1 contains S_3. Furthermore, it is also possible to show that checking the containment of a subscription within another requires polynomial time [10].

Quality Notification Service Running Examples. In order to illustrate the defined language and the notions of matching and containment, we now present some examples based on the global schema of the running example.

Table 3
Examples of subscription to Quality Notification Service

#	**Subscription**
S_1	$\langle Enterprise/Holder, \perp, \perp\rangle$
S_2	$\langle Enterprise/Holder, \perp, Name.Accuracy \leq$ `medium` $\rangle$
S_3	$\langle Enterprise/Holder, FiscalCode =$ "$MCLMSM73H17H501F''$, $\ldots$
	$\ldots Name.Accuracy \leq$ `medium` $\rangle$
S_4	$\langle Enterprise/Holder : \{INPS, INAIL\}, FiscalCode =$ "$MCLMSM73H17H501F''$, $\ldots$
	$\ldots Name.Accuracy \leq$ `medium` $\rangle$

Consider the set of example subscriptions in Table 3. Subscription S_1 refers to each quality change notifications for data objects belonging to the class `Holder` (son of class `Enterprise`), from any data source in the system. The following subscriptions limit the set of interesting quality change notifications. In particular, S_2 subscribes for notifications of quality changes in the `Name` attribute of objects of class `Holder`, when the accuracy of this attribute falls under or is equal to `medium`. Subscription S_3 further restricts the set of data objects of interest to holders whose fiscal code is equal to a specific value. Finally, subscription S_4 is the same as the previous one, but involves only data objects stored in two specific sources.

It is easy to see that subscription S_2 is contained in subscription S_1 and contains S_3 that in turn contains S_4. Therefore S_1 contains all others.

Concerning matching, a valid quality change event generated by a Quality Factory might be:

[10] More precisely, if the complexity of checking if a constraint is contained within another is constant, being n the sum of the number of constraints contained in two subscriptions, then the check costs $O(n^2)$.

$$\langle Enterprise/Holder, Name = \text{“}M.\ Mecalla\text{”} Address = \\ \text{“}Via\ dei\ Gracchi\ 71,\ Roma\text{”} FiscalCode = \\ \text{“}MCLMSM73H17H501F\text{”},\ \{\langle \texttt{name.Accuracy} = low \rangle\}\ \rangle$$

The afore event is received by Quality Notification Service that notifies all interested users. The notification has the same form of the event pushed by QF. In the case of the example subscriptions given in Table 3, it is easy to see that the notification matches S_1, S_2, and S_3. Matching with subscription S_4 depends on the organization in which the quality change occurs.

5.2 Overview of Quality Notification Service Implementation

The Quality Notification Service is implemented and deployed as a distributed service. Each organization hosts an independent instance of the Quality Notification Service that adopts the architecture described in this section. A Quality Notification Service instance accepts subscriptions only from users inside the organization it resides in (i.e. its *local users*) and receives notifications from the local Quality Factory. Quality Notification Service instances communicate among each other in a peer-to-peer fashion. That is, a Quality Notification Service [11] acts as a *producer* of information when it has to propagate to other Quality Notification Services the notifications produced by the Quality Factory of its organization. On the other hand, QNS acts as a *consumer* of information when it receives notifications produced in other organizations' Quality Notification Services, to be dispatched to its local users. Clearly, a Quality Notification Service can be a producer only for data classes of the global schema for which its organization is a source.

The design of the internal architecture of the Quality Notification Service has to face two main problems: *(i)* to cope with the heterogeneity of platforms and network infrastructures building the system and *(ii)* to scale to the large size we can expect in a CIS context.

Concerning heterogeneity, a standard technological solution, i.e. Web Services, is exploited for the communications among Quality Notification Service instances in order to achieve independence from the specific technological platform of each organization.

Concerning scale, the high number of possible users, each possibly issuing several subscriptions, and the high rate of notifications possibly produced by the Quality Factories are factors that could impact the performance of the system, in terms of computational resources and network bandwidth. In

[11] In the following, for simplicity we refer to a Quality Notification Service instance as “a Quality Notification Service ” whenever this is not confusing.

particular, the high number of subscriptions that could be issued throughout the whole system, imposes a high memory and computational overhead for storing and matching each notification, while possible frequent notifications may generate a large network traffic.

Both problems of heterogeneity and scale are faced through a layered architecture, in which a layer (namely, the *External Diffusion Layer, EDL*) deals with diffusing subscriptions and notifications among all organizations, while another layer (namely, the *Internal Interface Layer, IIL*) deals with handling subscriptions and notifications for all local users. The subscriptions of local users in an organization are stored inside the IIL. That is, each Quality Notification Service instance "knows" only its local users' subscriptions but has only a *partial* knowledge about the subscriptions issued by users of other organizations. Knowledge about non-local subscriptions is propagated *selectively* by exploiting containment relations, as we detail in the following, limiting memory consumption and network traffic.

In the following we describe in further detail how the Quality Notification Service works by presenting two solutions, namely Merge Subscriptions and Diffusion Trees, exploited in the layered architecture of the Quality Notification Service in order to deal with scalability.

Merge Subscriptions. Under a complete absence of "knowledge" about subscriptions, when a Quality Notification Service acts as a producer, each notification should be blindly flooded to all other Quality Notification Services, in order to surely reach all interested users. This would cause much unnecessary network traffic as each notification would be received also by Quality Notification Services with no interested local users.

To avoid this, the EDL of the Quality Notification Service in the generic organization O_i maintains a particular subscription, namely the *merge subscription*, denoted as $\mathcal{M}_i$. $\mathcal{M}_i$ contains all and only the subscriptions of all the local users of O_i and is computed from the union of the latter. For example the merge of subscriptions S_2, S_3 and S_4 in the examples of Table 3 is the subscription S_2 itself. $\mathcal{M}_i$ allows to receive all and only notifications matching some subscriptions issued by some users of O_i.

The merge subscription of each consumer organization is sent to all the producers of notifications possibly matching the merge subscription itself. On the consumer side, the schema mapping is exploited to determine the set of possible producers for a subscription. This allows for further reducing the Quality Notification Service instances involved in the communication with a given consumer only to those that can actually generate *interesting* notifications.

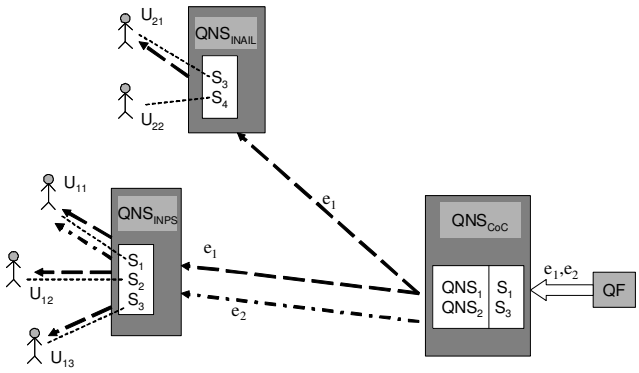


Fig. 14. Exploiting Merge Subscriptions

Upon a new notification, a Quality Notification Service acting as a producer has to check whether the notification matches the merge subscription of some other Quality Notification Services. Let us show this with the example depicted in Figure 14. Consider a producer Quality Notification Service, QNS_{CoC}, and two consumers QNS_{INPS} and QNS_{INAIL}. Suppose QNS_{INPS} manages three users, respectively interested in subscriptions S_1, S_2 and S_3 of example in Table 3, while QNS_{INAIL} manages two users, interested in subscription S_3 and S_4. The merge subscriptions for QNS_{INPS} and QNS_{INAIL} are respectively subscription S_1 and S_3. These are stored in QNS_{CoC}. Now consider the two following quality changes events generated in CoC for the dataclass class *holder*:

$$e1: \langle Enterprise/Holder, Name = \text{“}M.\ Mecalla\text{”} Address = \text{“}Via\ dei\ Gracchi\ 71, Roma\text{”} FiscalCode = \text{“}MCLMSM73H17H501F\text{”},\ \{\langle \texttt{name.Accuracy} = low \rangle\}\ \rangle$$

$$e2: \langle Enterprise/Holder, Name = \text{“}M.\ Scannapieco\text{”} Address = \text{“}Via\ Mazzini\ 27,\ Pagani\ (SA)\text{”} FiscalCode = \text{“}SCNMNC74S70G230T\text{”},\ \{\langle \texttt{name.Accuracy} = medium \rangle\}\ \rangle$$

$e1$ matches S_1, S_2 and S_3, while $e2$ matches only S_1. Due to the afore propagation of merge subscriptions, QNS_{CoC} can avoid the sending of $e2$ to QNS_{INAIL} as it can state from the merge subscriptions that QNS_{INAIL} has no users interested in $e2$. On the other hand, $e1$ is sent to both consumers Quality Notification Services, as they all have users interested in it. However, note that $e2$ is forwarded only to user U_{11} in QNS_{INPS} and $e1$ is not forwarded to user U_{22} in QNS_{INAIL}. However, the additional matching check required to determine the users of a notification is handled by consumer Quality Notification Services, independently from the producer.

Let us finally point out that also subscription traffic is reduced due to the use of merge subscriptions. Indeed, only changes in subscription that provoke a change in a merge subscription of an organization have to be propagated to producers. For example, if a new user in QNS_1 subscribes to subscription

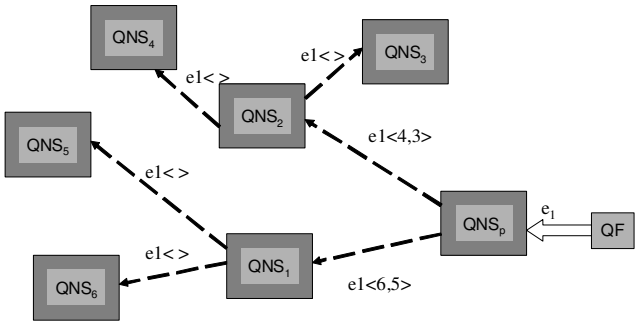


Fig. 15. Diffusion Tree Example

S_2, this is not communicated to QNS_p because it does not affect the merge subscription, that remains S_1.

Overall, the use of merge subscriptions requires an additional computational step when a new subscription and a new notification are issued. As valuable counterpart, the step allows to let flow between consumer and producer organizations only the *necessary* subscriptions and notifications, cutting off all the "useless" inter-organization network traffic.

Nevertheless, even the necessary traffic generated by notifications could be enough to overload some Quality Notification Services. The following section describes a mean to reduce the probability of this event.

Diffusion Trees. If a large number of Quality Notification Service consumers is interested in notifications coming from the same producerQuality Notification Service, the latter could be forced to open a large number of concurrent network connections to reach all of them. In the presence of a high rate of notifications, the scalability of the system could result compromised. To reduce the impact of this phenomena, each producer Quality Notification Service, once determined all the consumers of a notification, does not directly send it to them, but it rather calculates a diffusion tree, rooted at the producer and spanning all the consumers. The fan out of each node of the three is evaluated on the basis of information about the network connectivity capabilities of each network node. Each notification is thus sent from a node of the tree to its sons along with the information about the subtree of further consumers interested in the notification (see Figure 15). This approach enables (i) to limit the number of concurrent network connections managed by a producer and (ii) to distribute the load of notification propagation among all consumers according to their capabilities. Therefore, Quality Notification Services at intermediate levels of the tree act as forwarders of the notification for the lower levels. That is, upon receiving a notification, they also send it again to the Quality Notification Services at the immediate lower level in the tree.

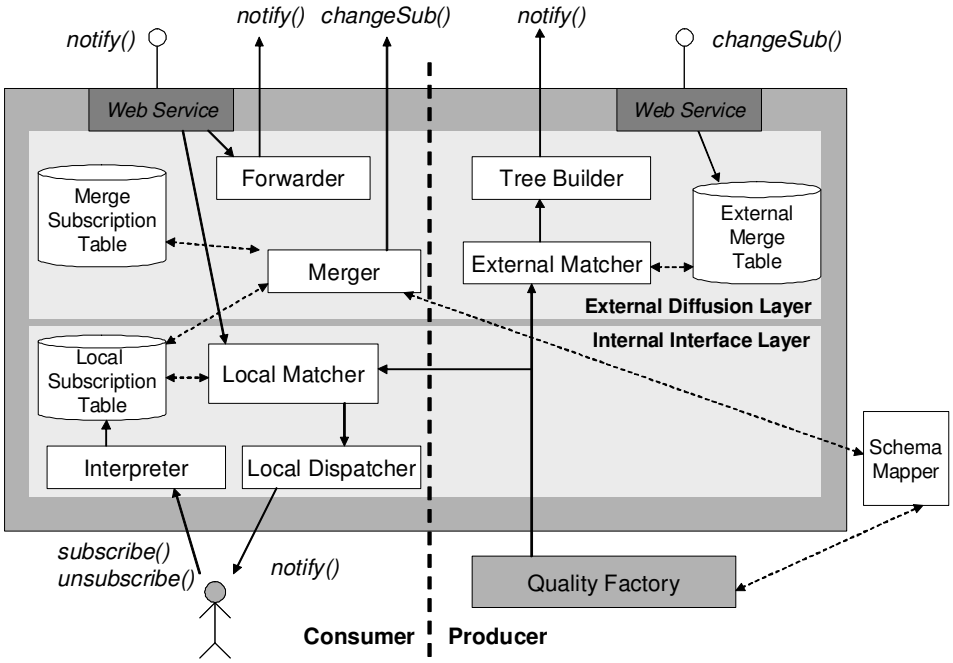


Fig. 16. Quality Notification Service

5.3 Quality Notification Service Internal Architecture

Figure 16 depicts the internal architecture of a generic Quality Notification Service instance, and shows also the internal fine-grained functional components of each of the Quality Notification Service layers. In the remainder of this section we detail these components.

Internal Interface Layer (IIL). The IIL consumer side comprises the following components:

- *Interpreter*: receives subscriptions from local users, interprets the subscription language, and stores them into the *Local Subscription Table*, i.e. a data structure storing all the subscriptions of all local users of an organization;
- *Local Matcher*: receives quality change notifications from the quality factory and from the EDL, and then matches them against the Local Subscription Table in order to return notifications to interested local users;
- *Local Dispatcher*: implements the dispatching mechanisms used to actually send notifications to local users.

The implementation of the subscription and dispatching mechanisms depend on the specific internal communication infrastructure of each organization, i.e. the Interpreted and the Local Dispatcher can be customized for each Quality Notification Service instance taking into account specific technological issues (e.g. pre-existence of a publish/subscribe middleware);

On the producer side, the IIL receives quality change events from the local Quality Factory of the organization. The data is first checked inside the organization via the Local Matcher to be dispatched to interested local users. The data is also transferred to components of the EDL to propagate it outside the organization.

External Diffusion Layer (EDL). As aforementioned, this layer is responsible for inter-organization notification diffusion. We explained in section 5.2 that the communication among different organization is based on Web Services technology. This means that each EDL instance exposes a Web Service interface, containing two methods, namely `notify()` (on the consumer side) and `changeSub()` (on the producer side), that are used to receive from other Quality Notification Services a notification and a change of merge subscription, respectively.

The producer side of the EDL of a Quality Notification Service instance comprises the following components:

- *External Matcher*: it checks the Quality Notification Service instances to which a notification has to be sent, by matching the notification against the merge subscriptions of other Quality Notification Services. The *External Merge Table* data structure maintains the merge subscriptions of consumer organizations, received from the Merger component of their Quality Notification Service (see below). As a consequence, such organizations are all and only the possible consumers (i.e. they have at least one subscribed user) for data classes for which O_i is a producer. Figure 14 shows the External Merge Table for QNS_{CoC}.
- *Tree Builder*: Determines the actual diffusion path followed by notifications and starts the notification diffusion. Notifications are propagated by invoking the `notify()` method on consumer Quality Notification Services. Each invocation contains also the specification of the remaining parts of the diffusion tree to which the notification has to be forwarded by the receiver, as shown in Figure 15.

The consumer side of the EDL of a Quality Notification Service instance comprises the following components:

- *Merger*: determines the merge subscription for the organization, when a subscribe/unsubscribe request occurs. A data structure, namely the *Merge Subscriptions Table*, maintains the merge subscription of the organization as seen by each producer Quality Notification Service [12]. The Merger sends

[12] Let us remark that a merge subscription is, in the general case, obtained from the union of a set of user subscriptions that do not satisfy containment. Therefore a merge subscription can be represented as a set of user subscriptions, which are

updates to producers whenever the merge subscription changes. As aforementioned, producers are identified using the *Schema Mapper*.
- *Forwarder*: responsible for sending the notification to all the Quality Notification Services at lower levels of the diffusion tree, by invoking their `notify()` method.

Let us conclude this section by pointing out that in essence Quality Notification Service is a *content-based* publish/subscribe system, i.e. users can subscribe by specifying information about the content of the notifications they are interested in. Implementing content-based pub/sub systems in large scale distributed systems is a hard task, in particular when aiming to implement efficient *matching* and *routing* algorithms (see Section 6 for further details). However, in this work the design of solutions for facing these issues has been simplified by taking into account the existence of other items of the DaQuinCIS framework (in particular, of the D^2Q model, of the Schema Mapper, and of the Quality Factory).

6 Related Work

Data Quality has been traditionally investigated in the context of single information systems. Only recently, there is a growing attention towards data quality issues in the context of multiple and heterogeneous information systems [28,3,29].

In cooperative scenarios, the main data quality issues regard [3]: *(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)* issue, some of the results already achieved for traditional systems can be borrowed, e.g., [30,31].

Methods and techniques for exchanging quality information *(ii)* have been only partially addressed in the literature. In [28], an algorithm to perform query planning based on the evaluation of data sources' quality, specific queries' quality and query results' quality is described. The approach of [28] is different from our approach mainly because we assume to have quality metadata associated to each quality value: this gives us the possibility of more accurate answers in selecting data on the basis of quality. This is specifically important for CIS's, in which the requirements on data to be exchanged are very strict.

stored in the Merge Subscription Table.

When considering the issue of exchanging data and the associated quality, a model to export both data and quality data needs to be defined. Some conceptual models to associate quality information to data have been proposed that include an extension of the Entity-Relationship model [32], and a data warehouse conceptual model with quality features described through the Description Logic formalism [31]. 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 [33], 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 metadata. 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 [33].

The heterogeneity *(iv)* issues are taken into account by data integration works. Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data [8]. As described in [34], when performing data integration two different types of conflicts may arise: *semantic conflicts*, due to heterogeneous source models, and *instance-level conflicts*, due to what happens when sources record inconsistent values on the same objects. The Data Quality Broker described in this paper is a system solving instance-level conflicts. Other notable examples of data integration systems within the same category are AURORA [34] and the system described in [35]. AURORA supports conflict tolerant queries, i.e. it provides a dynamic mechanism to resolve conflicts by means of defined conflict resolution functions. The system described in [35] describes how to solve both semantic and instance-level conflicts. The proposed solution is based on a multidatabase query language, called FraQL, which is an extension of SQL with conflict resolution mechanisms. Similarly to both such systems, the Data Quality Broker supports dynamic conflict resolution, but differently from them the Data Quality Broker relies onto quality metadata for solving instance-level conflicts. A system that also takes into account metadata for instance-level conflict resolution is described in [36]. Such a system adopts the ideas of the Context Interchange framework [37]; therefore, context dependent and independent conflicts are distinguished and accordingly to this very specific direction, conversion rules are discovered on pairs of systems.

For what concerns publish/subscribe communication systems, research has focused on algorithms to face large-size scenarios, considering for example efficient information diffusion ([50,48,49]) or efficient matching in content-based systems ([26,43,41,42]), while other works take into account QoS parameters

such as reliable message delivery ([52]). Descriptions of practical experiences with pub/sub systems can be found in [51,53]. To the best of our knowledge, the Quality Notification Service is the first content-based pub/sub infrastructure specifically designed for a cooperative context exploiting a data integration approach. As a consequence, and differently from "generic" content-based pub/sub systems, Quality Notification Service is enabled to exploit other services such as the Schema Mapper to improve matching and routing of subscriptions and notifications.

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 architecture to support data quality management in CIS's has been proposed. The specific contribution of the paper are *(i)* a model for data and quality data exported by cooperating organizations, *(ii)* the design of a service for querying and improving quality data and, *(iii)* the design of a service notifying changes on quality data to interested organizations.

Up to now, the pessimistic mode of the Data Quality Broker has been implemented. We plan to investigate in future work the research issues described throughout the paper concerning the optimistic invocation mode. We are currently investigating techniques for query optimization based on both data quality parameters and QoS parameters. The optimization based on data quality parameters can be performed on the basis of statistics on quality values. As an example, statistic values could be calculated on current data exported by organizations (e.g., "the medium accuracy of citizens currently provided by organization $\mathcal{A}$ is high"). As another example, statistic values could be calculated on the basis of past performances of organizations in providing good quality data (e.g., "the accuracy of citizens provided by organization $\mathcal{A}$ in the last N queries is high "). In such a way, the number of possible query answers is reduced by eliminating those that may not be conform to quality requirements according to the statistical evaluation.

The complete development of a framework for data quality management in CIS's requires the solution of further issues.

An important aspect concerns the techniques to be used for quality dimension measurement. In both statistical and machine learning areas, some techniques could be usefully integrated in the proposed architecture. The general idea is to give to each data value a quality evaluation with a certain estimated

probability, instead of a deterministic quality evaluation. In this way, the task of assigning quality values to each data value could be considerably simplified.

Acknowledgement

This work has been carried out in the context of the DaQuinCIS project, therefore we would like to thanks all people involved in the project, especially Barbara Pernici, Pierluigi Plebani and Sara Tucci Piergiovanni.

Special thanks are due to Carlo Batini, Tiziana Catarci and Maurizio Lenzerini for their support and useful discussions.

Finally, special thanks are due to the students who effectively realized the DaQuinCIS prototype during their master thesis: Luca De Santis, Diego Milano, Valeria Mirabella and Gabriele Palmieri. Without them, all such a work would be not a reality.

References

[1] G. De Michelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopoulos, M. Papazoglou, K. Pohl, J. Schmidt, C. Woo, E. Yu, Cooperative Information Systems: A Manifesto, in: M. Papazoglou, G. Schlageter (Eds.), Cooperative Information Systems: Trends & Directions, Accademic-Press, 1997.

[2] C. Batini, M. Mecella, Enabling Italian *e*-Government Through a Cooperative Architecture, IEEE Computer 34 (2).

[3] P. Bertolazzi, 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.

[4] R. Wang, A Product Perspective on Total Data Quality Management, Communications of the ACM 41 (2).

[5] Y. Wand, R. Wang, Anchoring Data Quality Dimensions in Ontological Foundations, Communications of the ACM 39 (11).

[6] T. Redman, Data Quality for the Information Age, Artech House, 1996.

[7] C. Cappiello, C. Francalanci, B. Pernici, P. Plebani, 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.

[8] 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.

[9] P. Aimetti, C. Batini, C. Gagliardi, M. Scannapieco, The "Services To Enterprises" Project: an Experience of Data Quality Improvement in the Italian Public Administration, in: Experience Paper at the 7th International Conference on Information Quality (IQ'02), Boston, MA, USA, 2001.

[10] P. Hall, G. Dowling, Approximate String Matching, ACM Computing Surveys 12 (4).

[11] L. Pipino, Y. Lee, R. Wang, Data Quality Assessment, Communications of the ACM 45 (4).

[12] D. Ballou, R. Wang, H. Pazer, G. Tayi, Modeling Information Manufacturing Systems to Determine Information Product Quality, Management Science 44 (4).

[13] P. Missier, M. Scannapieco, C. Batini, Cooperative Architectures: Introducing Data Quality, Technical Report 14-2001, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza" (Roma, Italy, 2001).

[14] A. Tansell, R. Snodgrass, J. Clifford, S. Gadia, A. S. (eds.), Temporal Databases, Benjamin-Cummings, 1993.

[15] E. Codd, Relational Database: a Practical Foundation for Productivity (1981 ACM Turing Award Lecture, Communications of the ACM 25 (2).

[16] R. Bruni, A. Sassano, Errors Detection and Correction in Large Scale Data Collecting, in: Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis, Cascais, Portugal, 2001.

[17] J. Ullman, Information Integration Using Logical Views, in: Proceedings of the Sixth International Conference on Database Theory (ICDT '97), Delphi, Greece, 1997.

[18] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Simèon, XQuery 1.0: An XML Query Language, W3C Working Draft (November 2002).

[19] D. Milano, , Tesi di Laurea in Ingegneria Informatica, Università di Roma "La Sapienza", Facoltà di Ingegneria (2003 (in Italian) (the thesis is available by writing an e-mail to: `monscan@dis.uniroma1.it`)).

[20] M. Hernandez, S. Stolfo, Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem, Journal of Data Mining and Knowledge Discovery 1 (2).

[21] C. Hwang, K. Yoon, Multiple Attribute Decision Making, Lectures Notes in Economics and Mathematical Systems-Springer Verlag 186, 1981.

[22] T. Saaty, The Analytic Hierarchy Process, McGraw-Hill, 1980.

[23] A. Buchmann, F. Casati, L. Fiege, M. Hsu, M. Shan (Eds.), Proceedings of the 3nd VLDB International Workshop on Technologies for *e*-Services (VLDB-TES 2002), Hong Kong, China, 2002.

[24] T. Chandra, S. Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM (JACM) 43 (2) (1996) 225–267.

[25] M. Fischer, N. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM (JACM) 32 (2) (1985) 374–382.

[26] M. K. Aguilera and R. E. Strom and D. C. Sturman and M. Astley and T. D. Chandra, Matching Events in a Content-Based Subscription System, in: Symposium on Principles of Distributed Computing, 1999, pp. 53–61.

[27] F. Fabret and H. A. Jacobsen and F. Llirbat and J. Pereira and K. A. Ross and D. Shasha, Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems, SIGMOD Record (ACM Special Interest Group on Management of Data) 30 (2) (2001) 115–126.

[28] F. Naumann, U. Leser, J. Freytag, Quality-driven Integration of Heterogenous Information Systems, in: Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, UK, 1999.

[29] L. Berti-Equille, Quality-Extended Query Processing for Distributed Processing, in: Proceedings of the ICDT'03 International Workshop on Data Quality in Cooperative Information Systems (DQCIS'03), Siena, Italy, 2003.

[30] H. Galhardas, D. Florescu, D. Shasha, 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.

[31] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis (Eds.), Fundamentals of Data Warehouses, Springer Verlag, 1999.

[32] R. Wang, H. Kon, S. Madnick, Data Quality Requirements: Analysis and Modeling, in: Proceedings of the 9th International Conference on Data Engineering (ICDE '93), Vienna, Austria, 1993.

[33] G. Mihaila, L. Raschid, M. Vidal, Querying Quality of Data Metadata, in: Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98), Valencia, Spain, 1998.

[34] L. Yan, M. Ozsu, Conflict Tolerant Queries in AURORA, in: Proceedings of the Fourth International Conference on Cooperative Information Systems (CoopIS'99), Edinburgh, Scotland, UK, 1999.

[35] K. Sattler, S. Conrad, G. Saake, Interactive example-driven integration and reconciliation for accessing database integration, Information systems 28.

[36] K. Fan, H. Lu, S. Madnick, D. Cheung, Discovering and reconciling value conflicts for numerical data integration, Information systems 28.

[37] S. Bressan, C. H. Goh, K. Fynn, M. Jakobisiak, K. Hussein, H. Kon, T. Lee, S. Madnick, T. Pena, J. Qu, A. Shum, M. Siegel, The Context Interchange mediator prototype, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA, 1997.

[38] P.Th. Eugster, P. Felber, R. Guerraoui and A.M. Kermarrec, The Many Faces of Publish/Subscribe. Technical Report ID:2000104, EPFL, DSC, Jan 2001.

[39] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content Based Routing with Elvin4. In *Proceedings of AUUG2K, Canberra, Australia*, June 2000.

[40] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and Evaluation of a Wide-Area Notification Service. *ACM Transactions on Computer Systems*, 3(19):332–383, Aug 2001.

[41] A. Campailla, S. Chaki, E. M. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision. In *Proceedings of The International Conference on Software Engineering*, pages 443–452, 2001.

[42] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. Sturman. An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems. In *Proceedings of International Conference on Distributed Computing Systems*, 1999.

[43] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, 2000.

[44] B. Oki, M. Pfluegel, A. Siegel, and D. Skeen. The Information Bus - An Architecture for Extensive Distributed Systems. In *Proceedings of the 1993 ACM Symposium on Operating Systems Principles*, December 1993.

[45] Softwired Inc. ibus//messagebus. Web site - http://www.softwired-inc.com, 2002.

[46] R. E. Gruber, B. Krishnamurthy, and E. Panagosf. The architecture of the READY event notification service. In *Proceedings of The International Conference on Distributed Computing Systems, Workshop on Middleware*, Austin, Texas, 1999.

[47] A. Carzaniga, D. Rosenblum, and A. Wolf. Challenges for distributed event services: Scalability vs. Expressiveness. In *Engineering Distributed Objects '99*, Los Angeles CA, USA, May 1999.

[48] R. Preotiuc-Pietro, J. Pereira, F. Llirbat, F. Fabret, K. Ross, and D. Shasha. Publish/Subscribe on the Web at Extreme Speed. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Cairo, Egypt, 2000.

[49] A. I. T. Rowstron, A. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[50] P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kouznetsov. Lightweight Probabilistic Broadcast. In *Proceedings of The International Conference on Dependable Systems and Networks (DSN 2001)*, July 2001.

[51] K.P. Birman. "A Review of Experiences with Reliable Multicast,". *Software - Practice & Experiences*, vol. 9, no. 29, 1999.

[52] K.P. Birman. "The Process Group Approach to Reliable Distributed Computing,". *Communications of the ACM*, vol. 12, no. 36, 1993.

[53] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange Trading System. In *Proc. of the 27rd IEEE International Symposium on Fault-Tolerant Computing*, June 1997.

[54] R. Guerraoui and A. Schiper. Software-Based Replication for Fault Tolerance. *IEEE Computer* Special Issue on Fault Tolerance, vol. 30, April 1997, 1997.