DaQuinCIS : Exchanging and Improving
Data Quality in Cooperative Information Systems

Monica Scannapieco
Monica Scannapieco

DaQuinCIS : Exchanging and Improving Data Quality in Cooperative Information Systems

Thesis Committee
Prof. Carlo Batini (Advisor)
Prof. Tiziana Catarci (Advisor)
Prof. Paola Bertolazzi
Prof. Francesco Archetti

Reviewers
Prof. Louiqa Raschid
Prof. William E. Winkler
AUTHOR'S ADDRESS:
Monica Scannapieco
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma “La Sapienza”
Via Salaria 113, I-00198 Roma, Italy
and
Istituto di Analisi dei Sistemi ed Informatica
Viale Manzoni 30, I-00185 Roma, Italy
E-MAIL: monscan@dis.uniroma1.it
WWW: http://www.dis.uniroma1.it/~monscan/
## Contents

1 Introduction .......................... 1
   1.1 Data Quality and CISs .................. 2
   1.2 Contents and Main Results of the Thesis .... 3

2 Background and Related Work ... 9
   2.1 The Problem of Defining Data Quality ........ 9
       2.1.1 Proposals of Data Quality Dimensions ....... 10
       2.1.2 Data Quality vs. Software Quality ........ 18
   2.2 Data Quality Models .................. 22
       2.2.1 The Quality Entity Relationship Model [90] ... 23
       2.2.2 Extending the Relational Model: The Polygen Model [97] and the Attribute-based Model [99] ... 24
   2.3 Measuring & Improving Data Quality ........ 25
       2.3.1 Record Linkage Methods ................ 25
       2.3.2 Process-based Improvement .............. 28
   2.4 Cooperative Information Systems .......... 30
   2.5 Data Integration Systems .............. 32
       2.5.1 Basics on Data Integration Systems ....... 32
       2.5.2 Data Quality Driven Integration Systems .... 34
       2.5.3 Quality-driven source selection [64] ....... 36
       2.5.4 Data Quality Issues in Data Integration Systems .... 36

3 The DaQuinCIS Framework: the $D^2Q$ model and the Data Quality Broker .... 39
   3.1 Overview: The DaQuinCIS Framework .......... 39
       3.1.1 Running Example .................... 42
   3.2 The $D^2Q$ Model ..................... 42
       3.2.1 Data Quality Dimensions ............ 43
       3.2.2 The $D^2Q$ Data Model .............. 46
       3.2.3 The $D^2Q$ Quality Model .......... 47
       3.2.4 The $D^2Q$ Model: Data + Quality .... 50
3.2.5 Implementation of the $D^2Q$ Data Model . . . . . . . . . 52
3.3 Querying the $D^2Q$ model with XQuery . . . . . . . . . . . 57
  3.3.1 The XML query language: XQuery . . . . . . . . . . . 57
  3.3.2 Quality Functions . . . . . . . . . . . . . . . . . . . . . 61
3.4 The Data Quality Broker . . . . . . . . . . . . . . . . . . . . 63
  3.4.1 Query Processing . . . . . . . . . . . . . . . . . . . . . 63
  3.4.2 Internal Architecture and Deployment . . . . . . . . . . 67
3.5 Query Engine: The Mediation Function of the Data Quality
  Broker . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
  3.5.1 Mapping Definition . . . . . . . . . . . . . . . . . . . . . 72
  3.5.2 Query Processing Steps . . . . . . . . . . . . . . . . . . 77
  3.5.3 Path Expression Extraction . . . . . . . . . . . . . . . . 78
  3.5.4 PE Pre-Processing . . . . . . . . . . . . . . . . . . . . . 81
  3.5.5 Translation . . . . . . . . . . . . . . . . . . . . . . . . . 84
  3.5.6 Framing . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
  3.5.7 Query Engine Implementation . . . . . . . . . . . . . . . 85
  3.5.8 Transport Engine . . . . . . . . . . . . . . . . . . . . . . 89

4 The DaQuinCIS Framework: The Record Matcher, The Rating
  Service and Other Modules . . . . . . . . . . . . . . . . . . . . 91
  4.1 The Record Matcher . . . . . . . . . . . . . . . . . . . . . . . 91
    4.1.1 Automatic choice of the matching key . . . . . . . . . . . 93
    4.1.2 Duplicate detection . . . . . . . . . . . . . . . . . . . . . 97
    4.1.3 Experiments and Results . . . . . . . . . . . . . . . . . 97
  4.2 Ensuring Trust in DaQuinCIS:
    the Rating Service . . . . . . . . . . . . . . . . . . . . . . . . 104
    4.2.1 DaQuinCIS as a Peer-to-Peer System . . . . . . . . . . . 105
    4.2.2 Trust Model: Basics . . . . . . . . . . . . . . . . . . . . 107
    4.2.3 Definition of a Trust Parameter for Source Units . . . . . 109
    4.2.4 Rating Service Design . . . . . . . . . . . . . . . . . . . 112
    4.2.5 Experimental Validation of the Model . . . . . . . . . . . 113
  4.3 The Other Modules: Quality Notification Service and Quality
    Factory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
    4.3.1 Quality Notification Service: Overview . . . . . . . . . . . 115
    4.3.2 Quality Factory: Overview . . . . . . . . . . . . . . . . 115

5 The DaQuinCIS Methodology: IP-UML+TDQM_CIS . . . . . . . . 117
  5.1 The IP-UML Methodology . . . . . . . . . . . . . . . . . . . . 118
    5.1.1 IP-MAP Framework Basics and UML Extension . . . . . . 119
    5.1.2 A Running Example . . . . . . . . . . . . . . . . . . . . . 121
    5.1.3 Data Quality Profile . . . . . . . . . . . . . . . . . . . . 122
    5.1.4 Data Analysis Model . . . . . . . . . . . . . . . . . . . . 122
Chapter 1

Introduction

Electronic data are one of the bases of our information and communication society; they are managed by business applications, governmental applications, diverse kind of applications on the Web. With electronic data so basic rooted to all facets of an information and communication society, it is not possible to ignore the importance of the “quality” of such data.

Quality of data is not simply their correctness. Let us consider the case of an address to deliver some mail: it can be correct, as having a street name and a street number specified, but if city is not included, it is useless for most applications. The example shows that quality of data also means completeness. In a similar way we could show that data quality means many other things, such as currency, interpretability, consistency and other “dimensions”, often depending from the context where data are used and also from specific users within a given context. It follows that data of good quality may be not easily obtained. Data quality is a complex concept defined by multiple dimensions and depending on many variables, some of which can be very subjective.

Despite this complexity, the crucial role of data requires us to face and solve data quality problems.

A recent report of the Data Warehousing Institute (TDWI) [33] states the follows:

... there is a significant gap between perception and reality regarding the quality of data in many organizations, and that current data quality problems cost U.S. businesses more than $600 billion a year.

For many years, research on data quality has been carried out by statistics researchers, most of them working in governmental agencies. Historically, they were the first to start investigating data quality. Since the results of their activity was strictly determined by the quality of data that they collected through questionnaires, interviews, etc. One of the needs they had was
to quantitatively define errors in collected records, in order to exactly define the reliability of statistical analysis. Furthermore, when errors exceeded acceptable thresholds, they also had the need to correct the data; to such a scope they developed edit/imputation methods [106].

More recently, computer scientists began investigating data quality issues, driven by the needs of modern information systems. Quality of data has often been poor, but when data were confined to single isolated contexts, it was easier to handle data quality problems, or even, in some cases, to ignore them. Modern information systems are often the result of multiple systems interacting with each other. Even when a modern information system is not connected to others, it is not a “closed” system anymore, since it typically has at least one window open over the world, namely the Web. This complicates the data quality problem: when data are exchanged, compared and shared, there is less tolerance for poor quality.

In the specific environments characterized by extensive data replication, high quality of data becomes a strict requirement. Federated databases and data warehouses are only some examples of environments characterized by data replication. Poor quality data are unacceptable obstacles to data integration in such contexts and constitute a problem that must be solved. Let us also note that other contexts of partially or totally replicated data sources that do not require any data integration architecture are also deeply affected by data quality problems. As an example, virtual enterprises are focused on business processes involving different geographically distributed enterprises; though data locally stored may be not integrated, they still have the problem of having reconciled copies of common data. For instance, customer data must be the same in databases local to each enterprise in the virtual community.

1.1 Data Quality and CISs

We define Cooperative Information Systems (CISs) as all distributed and heterogeneous information systems that cooperate by sharing information, constraints, and goals [20, 67]. CISs include data integration systems as well as systems that share common information while not explicitly integrating data.

In the literature, CISs are seen as the “next generation information systems” [28], that “... will manage or have access to large amounts of information and computing services ... Information and services will be available in many forms through legacy and new information repositories that support a host of information services ...”.

We argue that quality of data is a necessary requirement for a CIS, i.e. CISs need data quality. First, a system in the CIS will not easily exchange data with another system without knowledge of the quality of such data, and
cooperation becomes difficult without data exchanges. Second, when poor quality data are exchanged, there is a progressive deterioration of data stored in the whole CIS. Indeed, uncontrolled exchanges of low quality data cause a diffusion of such data throughout the system, thus lowering the quality of each data asset in the CIS. Third, when a CIS is a data integration system, data integration itself cannot be performed if data quality problems are not fixed. For example, results of queries executed over local sources must be reconciled and merged, and quality problems resulting from a comparison of results need to be solved in order to return an answer to a global query [18].

On the other hand, data replication in the CIS can be exploited for improving data quality, i.e. \textit{data quality needs CISs}. Though it is possible to enact quality improvement actions by simply looking at single sources, there are some limitations that cannot be overcome by having only one copy of data to look at. For instance, accuracy improvement can be performed by having syntactic dictionaries as references; therefore, values for a field name could be quite easily syntactically checked. Nevertheless, things can become more difficult if field values cannot be compared with reference dictionaries; for instance, this is often the case of numerical data. Furthermore, even in the case in which syntactical checks can be applied, the semantic problem is not solved; for instance, if a date is syntactically correct, i.e., it respects a given format, but it is actually the wrong date, this is not easily detected by simply looking at one source. Instead, in CIS’s different copies of the same data are typically stored by multiple sources and can be compared in order to detect quality problems and possibly solve them.

1.2 Contents and Main Results of the Thesis

Starting from the issues discussed in the previous section, this thesis proposes the DaQuinCIS (Data Quality in Cooperative Information Systems) approach for supporting data quality in cooperative information systems.

Methods for data quality analysis and improvement have been focused on single information systems, in the past years. Specifically, two classes of methods have emerged, namely data-based methods (such as record matching) and process-based methods (such as process re-engineering methods). Instead, little attention has been paid on data quality in cooperative environments.

The DaQuinCIS approach has the principal objective of dealing with data quality in the domain of cooperative systems either under a methodological point of view or from an architectural one.

Although some of the data-based and process-based methods could be applied in such context, the lack of an integrated methodological framework that considers the characteristics of cooperative information systems actually
CHAPTER 1. INTRODUCTION

limits the possibility of a systematic and coordinated use of such methods.

From an architectural perspective, the DaQuinCIS approach investigates the services needed for improving data quality in cooperative environments and a platform for their implementations.

The main results of this thesis can be placed within the definition and design of the DaQuinCIS system and the DaQuinCIS methodology. The DaQuinCIS system is a suite of services deployed on a peer-to-peer platform in order to exchange and improve data quality in cooperative information systems. The DaQuinCIS methodology integrates an intra-organization methodology for data quality improvement with an inter-organization one. The specific contributions of the thesis are listed below:

- The definition of a graph-based data model, in order to represent (i) data to be exported in a CIS environment for being exchanged, (ii) a common set of data quality dimensions and (iii) the associations between such data and quality dimensions. Though existing in the literature models to represent data quality both at a conceptual and at a logical level, none of such models has the characteristics required to be suitable for a CIS environment. More specifically, the proposed model is semistructured, thus allowing to associate quality data to data in a more flexible way, with respect to other structured proposals. The semistructured nature of the proposed model is also well-suited to capture also semistructured data sources, such as XML data bases, that can be present in CIS’s in addition to structured relational data bases. Furthermore, the proposed model can be easily implemented by using XML related standards; in cooperative context in which the interoperability issues are particularly important, the usage of such standards becomes very relevant. The proposed data model is described in Section 3.2.

- The definition of a query mechanism to access quality data and specify data quality requirements. More specifically, we extend an XML query language, i.e. XQuery, in order to access quality dimension values defined for data according to the proposed model. Instead of defining a new query language, we have exploited the possibility of XQuery to have user-defined functions, thus enriching with quality features a language that is the best candidate to become a standard for XML querying. Further details on the data quality functions extending XQuery can be found in 3.3.

- The definition of a query processing technique in a data integration settings in which local and global schemas are expressed according to the proposed data model. Here two contributions can be identified. Firstly, we define a data quality based semantics for a data integrations system,
in which given a query on the global schema, the result of the query is the “best” quality result available in the CIS. Secondly, differently from what happens in a traditional relational setting, the concept of querying semistructured integrated data is a quite new field. Therefore, query processing steps that are consolidated in the relational setting, such as the unfolding phase with a Global As View mapping, needed to be rethought according to the semistructure nature of the adopted data model. Section 3.4 and Section 3.5 describe the detail of such issues.

- The definition of a record matching algorithm, extending a well-known algorithm proposed in the literature, in order to allow an automatic choice of a matching key in presence of quality data. Instead of relying on “key designers”, our idea is to rely on quality data associated to exported data values, in order to choose a “good” key for matching. In CIS’s, the key design problem is very complex because of the multiple and different data sources, therefore an automatic support is necessary. The method is also applicable in any case in which quality metadata are available on data to be matched. Experimental tests show the effectiveness of the method by itself and compared with the original algorithm. The record matching algorithm is described in Section 4.1.

- The definition of a probabilistic model to trust sources with respect to provided data in a cooperative environment. Though models for trusting sources in peer-to-peer environments are available in the literature, as to our knowledge, no model has been already proposed that characterizes a source with respect to a provided data type. Instead, this issue is particularly relevant for CIS’s in which data sources can actually have different trust values for different data types. By distinguishing them according to this feature, we give a mean to individuate a subject that is responsible for a certain data category, i.e., the so-called data steward, as the data source having the highest trust level with respect to a specific data category. This comes very useful when designing processes for quality improvement. The model has been experimentally validated and is described in Section 4.2.

- The definition of a methodology to improve data quality in CIS’s. We first recognized the need of proposing a methodology for quality improvement inside each cooperating organization. To such a scope, we do not set aside the results already published in the data quality literature; instead, we chose to rely on one of them by revisiting it according to the requirements that a cooperative setting could ask to single organizations. Then, we also propose a methodology for data quality improvement to be applied to a CIS as whole, and thus from the outside of a cooperating
CHAPTER 1. INTRODUCTION

organization.

The DaQuinCIS methodology results from the integration of these two. As to now, the DaQuinCIS methodology has been partially validated in a real setting; some further experimentations are running to complete the validation process. The methodology is fully described in Section 5.3.

The overall design and implementation of the DaQuinCIS system as a peer-to-peer platform. The design of the DaQuinCIS system is a result of this thesis because very little attention has been payed in the literature to data quality in cooperative context. According to our knowledge, no previous proposal exists for a platform supporting quality-based querying and above all improvement based on replicated data comparisons. An overview of the DaQuinCIS framework design and of the main functionalities of the system is provided in Section 3.1 and in Section 3.4.

Besides the results of the thesis, we point out some hypotheses on which our work relies.

Firstly, we suppose that cooperating organizations internally perform a data quality assessment in order to populate the model for exporting data and quality data with quality dimension values. We do not investigate such internal data quality assessment techniques.

Secondly, the focus of the data integration system included in the DaQuinCIS framework is on data quality access and improvement. Therefore, our data integration system has some simplifying hypothesis with respect to other more “traditional” data integration systems. For instance, we do not deal with integrity constraints.

Our assumptions, in particular that each organization is able and is willing to export quality values, is motivated by the cooperative scenarios that we experienced. Specifically, we considered as a real scenario for the proposed ideas the CIS of Italian public administrations. Indeed, some hypotheses, requirements and methodological suggestions were derived from experiences of the author of this thesis within the Italian e-Government initiative [9].

Furthermore, an Italian research project provided the necessary context to validate the ideas of the thesis; the project is DaQuinCIS - Methodologies and Tools for Data Quality inside Cooperative Information Systems (http://www.dis.uniroma1.it/~dq/).

Some of the results of this thesis have been published in national and international journals, international conferences and workshops; more specifically:

The data model and issues concerning quality querying and improving has been published in [60, 61, 84, 62, 87].
Publications concerning the specific modules of the DaQuinCIS framework are [12, 29, 57, 58, 23].

The proposed methodological issues have been published in [13, 85, 86].

Some works about the data quality definition problem are [24, 69].

Some data quality experiences in the Italian e-Government setting are published in [4, 15, 36, 14].

Beyond data quality in cooperative systems, some related research results concern data quality in Web Information Systems (WIS’s). Such results are about dynamic data quality dimensions, and how they can be attached to web-published data. They are not described in this thesis as quite far from its scope, but the interested reader can refer to [73, 74, 75].
Chapter 2

Background and Related Work

This chapter describes the related work of this thesis, also providing some background concepts. More specifically, the following topics are discussed:

- Data quality definition and sets of dimensions characterizing data quality, described in Section 2.1.1.
- Data quality models, described in Section 2.2.
- Methods for measuring and improving data quality, both data-based, described in Section 2.3.1, and process-based, described in Section 2.3.2.
- Cooperative information systems’s generalities, illustrated in Section 2.4.
- Data integration systems’s basics and data quality relevant results achieved in the data integration area, discussed in Section 2.5.

2.1 The Problem of Defining Data Quality

Different definitions for data quality have been proposed. Generally speaking, data quality can be defined as “fitness for use” [101], and as “the distance between the data views presented by an information system and the same data in the real world” [72, 95]. The former definition emphasizes the subjective nature of data quality, whereas the latter is an “operational” definition, although defining data quality on the basis of comparisons with the real world is a very difficult task.

In this thesis, we consider data quality as a multidimensional concept and define it as a set of “dimensions” or “characteristics”. In this section, we discuss the proposals for data quality dimensions and the relationships between data quality and software quality.
2.1.1 Proposals of Data Quality Dimensions

In [24], we compared some proposals for data quality dimensions in order to point out analogies and differences. More specifically, in 1995, a survey on data quality research was published [100]; since then, many other proposals have been developed; among them we have selected six ones that are representative of different contexts.

We introduce them in the following:

- **WandWang96** [95]. It is based on a formal model for information systems. Data quality dimensions are defined by considering mapping functions from the real world to an information system. As an example, inaccuracy of data means that the information system represents a real world state different from the one that should have been represented. As another example, the completeness dimension is defined as a missing mapping from real world states to the information system states. A total of 5 dimensions are proposed: accuracy, completeness, consistency, timeliness, and reliability.

- **WangStrong96** [101]. The proposal derives from an empirical study. Data quality dimensions have been selected by interviewing data consumers. Starting from 179 data quality dimensions, the authors selected 15 different dimensions.

- **Redman96** [80]. The proposal groups data quality dimensions into three categories, corresponding to the conceptual view of data, the data values and the data format respectively. 5 dimensions are proposed for the conceptual view, 4 dimensions for the data values and 8 dimensions for the data format.

- **Jarke99** [51]. The proposal was made in the context of the European Research Project DWQ, Foundations of Data Warehouse Quality. The overall project objective is to guide data warehouse design activities. In this context, specific data quality dimensions are proposed. The dimensions are classified according to the roles of users in a data warehouse environment, namely: 6 dimensions for Design and Administration Quality, 6 dimensions for Software Implementation Quality, 5 dimensions for Data Usage Quality and 5 dimensions for Data Stored Quality.

- **Bovee01** [55]. Following the concept of data quality as “fitness for use”, the proposal includes 4 dimensions (with some sub-dimensions). Data are “fit for use” whenever a user: 1) is able to get information (Accessibility); 2) is able to understand it (Interpretability); 3) finds it applicable to a specific domain and purpose of interest (Relevance); 4) believes it to be credible (Credibility).
2.1. THE PROBLEM OF DEFINING DATA QUALITY

Naumann02 [43]. The proposal defines quality dimensions specific for integrated Web Information Systems. It considers 4 categories for a total of 21 dimensions. The four categories are: content-related, concerning the actual data that are retrieved; technical, concerning aspects related to the source, the network and the user; intellectual, related to subjective aspects of the data source; instantiation-related, concerning the presentation of data.

Comparison of data quality dimension proposals

When comparing the different proposals for data quality dimension sets, we highlight, in Figure 2.1 and Figure 2.2, two types of correlations among them, namely:

- In Figure 2.1, we show how different proposals use the same name for dimensions with Different (D) or Similar (D%) meanings. The letter S is used to indicate same names and same meanings for a dimension in the different proposals; this is outlined in order to consider which proposals include the same dimensions.

- In Figure 2.2, we see how different names for dimensions with Similar (D%) or Same (S) meanings are used.

According to Figure 2.1, accuracy and completeness are the only dimensions defined by all proposals. Besides these two specific dimensions, consistency-related dimensions and time-related dimensions are also taken into account by all proposals. Specifically, consistency is typically considered at instance level (consistency dimension) or at format level (representational consistency). Time-related quality features are mainly caught by the timeliness dimension. Also interpretability is considered by most of the proposals, both at format and schema level. Each of the remaining dimensions is included only by a minority of proposals. In some cases there is a complete disagreement on a specific dimension definition, such as for reliability. For detailed comments on each row of the table shown in Figure 2.1, the reader can refer to the Appendix of [24].

In Figure 2.2, we show similarity between dimensions that are named differently in the various proposals:

- In Figure 2.2 a), clarity of definition as defined in Redman96 is similar (D%) to interpretability as defined in WangStrong96, Bovee01 and Naumann02.

- In Figure 2.2 b), accessibility as defined in WangStrong96, Jarke99 and Bovee01, i.e. how much data are available or quickly retrievable, is the same (S) as obtainability of values in Redman96.
### Figure 2.1: Correspondences among dimensions with same names. D% means similar meanings, S means same meaning.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Completeness</td>
<td>S</td>
<td>D</td>
<td>S</td>
<td>D%</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Consistency</td>
<td>S</td>
<td>D%</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Representational Consistency</td>
<td>S</td>
<td>S</td>
<td></td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timeliness</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Currency</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Volatility</td>
<td>S</td>
<td>S</td>
<td></td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interpretability</td>
<td>S</td>
<td>D%</td>
<td>D%</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Ease of Understanding/Understandability</td>
<td>S</td>
<td>S</td>
<td></td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>D</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credibility</td>
<td>D</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Believability</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reputation</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objectivity</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relevancy/Relevance</td>
<td>S</td>
<td>S</td>
<td>D%</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Security</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value-added</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concise representation</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appropriate amount of data</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portability</td>
<td>D</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Responsiveness/Response Time</td>
<td>S</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.1. THE PROBLEM OF DEFINING DATA QUALITY

- In Figure 2.2 c), correctness as defined in Jarke99, i.e. proper comprehension of the entities of the real world, is the same (S) as comprehensiveness in Redman96.

- In Figure 2.2 d), minimality as defined in Jarke99, i.e. the degree up to which undesired redundancy is avoided, is the same (S) as minimum redundancy as defined in Redman96.

On the basis of the correlations in Figure 2.1 and Figure 2.2, it is possible to sketch a basic definition for data quality, i.e. a definition that includes features considered by the majority of the proposals.

We define data quality as: a set of dimensions including accuracy, completeness, consistency (at format level and at instance level), timeliness, interpretability and accessibility.

In fact, Figure 2.1 shows that accuracy, completeness, consistency, timeliness and interpretability are dimensions shared by most of the proposals. Combining Figure 2.1 and Figure 2.2, also accessibility has to be included in the basic set. The reader should notice that for each proposal only the dimensions that are shared by at least another proposal have been considered. As an example, the list of Jarke99’s dimensions includes six further dimensions besides the ones shown in Figure 2.1, that are not shared by any of the other proposals.

In summary, though there are several different dimensions in the various proposals, it is possible to single out a few of them that basically define the concept of data quality. All other dimensions included in the proposals either capture secondary features or are more context-dependent (i.e. very specific). This latter case will be deeply discussed in the next section.

A Classification of the Proposals: When Using Which Proposal

Though having provided a basic definition for data quality in Section 2.1.1, there are many cases in which more specific sets of dimensions are needed. The aim of this section is to classify the proposals for data quality dimensions in order to have some methodological suggestions guiding the choice of the best proposal according to one’s own requirements. We classify the proposals described in Section 2.1.1 according to four features:

- Approach to dimensions definition, i.e., what is the process followed to define the set of dimensions. We consider three types of possible approaches to dimension definition: Theoretical, Empirical and Intuitive. A theoretical approach means that a formal model or theory is proposed in order to define or justify the proposed dimensions. An empirical approach means that the set of dimensions is constructed starting from
Figure 2.2: Correspondences among dimensions with different names.
2.1. **THE PROBLEM OF DEFINING DATA QUALITY**

experiments, interviews and questionnaires. In the intuitive approach, dimensions are simply defined according to common sense.

- **Modeling view on data**, i.e., which data perspective has been taken in order to define quality dimensions. The considered perspectives are three: Schema, Instance and Format. Schema refers to the intensional view of data, such as the definition of a table in the relational model; the instance view is related to the extension of data, i.e. to actual values; the format view is related to how data are represented.

- **Measurement view on data**, i.e., how data are analyzed in order to evaluate their quality. Three different measurement views can be considered: Process, System and Product. The process view considers how the processes that produce data affect their quality. As an example, it is possible to consider the way in which a data entry process is carried on and how it affects the quality of the stored data. The system view is related to the consideration of the whole information system as influencing the quality of data. As an example, if a distributed information system is considered, some specific attention should be paid to time-related dimensions and to how their values are affected by data exchanges among different nodes. Instead, the product view is specifically related to data and to the user perception of their quality. More specifically, this view considers information as a product to be delivered to consumers and typically includes subjective dimensions like interpretability or understandability, which need to be evaluated by the final consumer of the information.

- **Context dependence**, i.e., if the proposal is tailored to a specific application context or is general purpose. The possible values of this classification variable are simply Yes or Not.

In Figures 2.3 a) and 2.3 b), the positions of the various proposals according to the described variables are shown. 2.3 a) shows the positions of each proposal in the plan Approach to Dimension Definition-Modeling View on Data. Notice that the only proposal covering all modeling views on data is Redman96. Jarke99 is specific for the data warehouse context, and Naumann02, for web integration systems. Figure 2.3 b) shows the position of each proposal in the plan Approach to Dimension Definition-Measurement View on Data. Notice that no proposal covers all measurement views on data.

The various approaches to dimension definition give useful information for a good choice of the best set of quality dimensions. An intuitive proposal may be sufficient if a general introduction to the data quality problem is required. Whereas, when data quality information has to guide strategic decisions, an empirical or theoretical approach may be more adequate.
Figure 2.3: A classification of dimensions proposals.
2.1. THE PROBLEM OF DEFINING DATA QUALITY

The different modeling views on data also help in focusing on the right abstraction level to be adopted. As an example, if unstructured information is considered, such as free text documents, no proposal concentrating on schemas may be useful. As another example, when considering images, a proposal specifically concerning format may be most appropriate.

With reference to measurement views, the process view allows to focus on quality improvement inside an organization, by detecting the key causes of poor quality. Organizational decisions may be taken, for example concerning the enactment of a Business Process Reengineering (BPR) activity. The system view becomes very important when a particular information system supports a given enterprise or organization. The product view is especially important when data are the real product of an enterprise, such as in public administrations that have as a main task to manage data about citizens and enterprises.

Finally, the context dependence is of great importance. Indeed, if a specific proposal fits the one’s own context, it is undoubtedly the right one to choose.

As an example of how to combine the different variables in order to choose a specific proposal of dimensions, let us consider the Italian e-Government scenario. Current Italian e-Government projects aim at creating a cooperative information system among public administrations. Autonomous administrations must be able to cooperate with other administrations since they do not have complete control over data and services needed to reach their own goals. In such a scenario, if an administration has to request data from another, it would like to be guaranteed about the quality of the data provided by this latter. An assessment of the quality provided by each administration is thus necessary in order to enable data exchanges among cooperating organizations.

Let us consider some examples of quality requirements that drive the choice of quality dimensions:

- Specific focus on data values, as they are the ones actually exchanged among the different administrations.
- Focus on data as a product to be delivered by the nationwide CIS to citizens and enterprises.
- Rigorous process to be followed in dimension definition.

According to such requirements a good set of dimensions can be provided by WangStrong96. In fact, this proposal focuses on instance view of data, thus addressing the requirement related to data values, and on a measurement view of product type. The empirical approach to dimension definition is also adopted thus addressing the need of a rigorous process in dimension definition. The fact that WangStrong96 also includes a measurement view of
data of process type (see Figure 2.3 b) makes this proposal suitable even for an improvement activity that could be engaged by single administration on their own data. As an example, an administration may find out that some typographical errors in the data it disseminates are due to the lack of an automatic data entry process. The solution may be a reengineering of the data entry process in which data are automatically loaded, for example from XML files sent by a different administration.

The proposed classification aims at guiding the designer in choosing data quality dimensions that fits his/her application needs. It is possible (and it is also very probable) that no proposal matches exactly one’s requirements. In such cases, the role of the classification is to help in pointing out the specific deficiencies of each proposal. Let us consider again our example of the Italian CIS among public administrations. CIS’s are information systems with a lot of specific characteristics; this suggests the usefulness of a set of dimensions also taking into account a system perspective. Let us suppose that the Department of Finance asks a City Council for some data related to a citizen’s family composition, in order to enact a tax assessment. Such data may arrive late and may cause the tax assessment process to be postponed because of a lack of system availability, a need of secure transmission, delays of the networks etc. Therefore, besides the dimensions strictly related to the quality of data, some other dimensions might need to be considered as directly impacting the quality of the data as perceived by the final consumer. In our example, the timeliness of the data arrived at the Department of Finance is affected by system quality problems. WangStrong96 does not allow for considering a measurement view on data of system type, so the choice of this proposal for the Italian CIS should be integrated with some specific dimensions explicitly taking into account system characteristics.

2.1.2 Data Quality vs. Software Quality

In the computer science field, when the term quality is used, it is often related to software quality. Data quality is instead a concept that still needs to be introduced and motivated. In [69], we fix the problem of the relationships between data quality and software quality. Indeed, it is intuitive that some kind of relationship should exist: software manipulates data and some quality attributes that are associated to software may derive from the quality of such data. Specifically, we establish a correspondence between:

- the international standard for software quality ISO/IEC 9126 [2];
- the set of quality dimensions proposed by Thomas Redman [80].
The Standard ISO/IEC 9126

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) published the 9126 as an international standard [2].

Such a standard defines the quality of a software product as a set of six characteristics, namely: functionality, reliability, usability, efficiency, maintainability and portability. The characteristics are further specified by 27 sub-characteristics that are shown in Figure 2.4.

A further classification that is proposed in the standard ISO/IEC 9126 distinguishes internal quality, external quality and usage quality. Internal quality is the quality of software evaluated by simply looking at the software product. External quality is related to the quality of software when executed. Usage quality is related to the user’s ability to effectively use the software product. The characteristics and sub-characteristics shown in Figure 2.4 refer to the external and internal quality and are the ones considered for a comparison with data quality dimensions.
CHAPTER 2. BACKGROUND AND RELATED WORK

<table>
<thead>
<tr>
<th>Data Quality Categories</th>
<th>Data Quality Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Value view</td>
<td>Accuracy, Completeness, Currency, Consistency.</td>
</tr>
<tr>
<td>Data Format view</td>
<td>Appropriateness, Interpretability, Portability, Format Precision, Format Flexibility, Ability to represent Null Values, Efficient Usage of Recording Media, Representation Consistency.</td>
</tr>
</tbody>
</table>

Table 2.1: Dimensions for values and format proposed in [80].

Redman’s data quality dimensions

In this section, we overview Redman’s proposal for data quality dimensions [80]. As discussed in Section 2.1.1, a large number of data quality dimensions sets has been proposed. For comparing with the ISO/IEC 9126 we have chosen to focus on Redman’s proposal as it was one of the most comprehensive. Notice also that it is the only proposal fully including the minimum set of dimensions defining data quality that has been identified in Section 2.1.1.

Redman groups data quality dimensions into three categories, corresponding to the conceptual view of data, the data values and the data format respectively. For comparing with the ISO/IEC 9126, we only consider the dimensions related to data values and data format. Indeed, the quality of conceptual view is not directly concerning software quality. The list of data value and data format dimensions of the Redman’s proposal is shown in Table 2.2.

Correspondences between ISO/IEC 9126 and data values dimensions. We identified only one semantic correspondence (which is also a name-based correspondence) between ISO/IEC 9126 and data values dimensions, namely between:

☑ the accuracy sub-characteristic of functionality defined as the set of attributes of software that bear on the provision of right or agreed results or effects;

☑ the accuracy defined as the proximity of a value \( v \) to a value \( v' \) considered as correct.

Correspondences between ISO/IEC 9126 and data format dimensions. In the following, we list the correspondences identified among ISO/IEC 9126 sub-characteristics (see Figure 2.4) and data format dimensions.
A sub-characteristics of functionality is related to a data format dimensions, namely:

- the sub-characteristic suitability is related to the data format dimension appropriateness.

The sub-characteristics of reliability have no correspondence with data format dimensions.

With respect to usability’s sub-characteristics, the following correspondences have been identified:

- the sub-characteristic understandability is related to the data format dimensions interpretability and portability;

- the sub-characteristic operability is related to the data format dimension appropriateness.

With reference to maintainability’s sub-characteristics, only one correspondence has been identified, namely:

- the sub-characteristic changeability is related to the data format dimension flexibility.

Among efficiency’s sub-characteristics, the following correspondence has been identified:

- the sub-characteristic resource behavior is related to the data format dimensions usage of recording media.

With respect to portability’s sub-characteristics, the following correspondence has been identified:

- the sub-characteristic adaptability is related to the data format dimension flexibility.

In Figure 2.5, all the identified correspondences are shown. The following observations can be made:

- the correspondence is not functional, i.e. a sub-characteristic is associated to more than one dimensions, namely: understandability is associated to both interpretability and portability. Moreover, not all ISO/IEC sub-characteristics have an associated quality dimension On the other hand, some dimensions are associated to more than one sub-characteristics; as an example, flexibility is associated to both changeability and adaptability.
Almost all format dimensions have a correspondence with ISO/IEC 9126 sub-characteristics. This is because the format of data defines a way to represent data, thus “operating” on data. In the same way, a software product “operates” on data in order to carry on its own functions.

Reliability is the only ISO/IEC 9126 characteristics that is no way related to data quality dimensions. This can be explained by considering that data are not reliable by themselves according to the ISO/IEC 9126 semantics, i.e. fault tolerant, mature etc. Instead, procedures that manage data should be reliable.

2.2 Data Quality Models

Data quality models have been proposed both at a conceptual level and at a logical level.
2.2. DATA QUALITY MODELS

At a conceptual level, an extension of the Entity Relationship model in order to include quality data has been proposed in [90]. This model is briefly described in Section 2.2.1.

At a logical level, two early attempts at extending the relational model have been proposed in [99, 97]. The main features of these proposals are described in Section 2.2.2.

2.2.1 The Quality Entity Relationship Model [90]

The Quality Entity Relationship model defines:

- a Data Quality Dimension entity, as \([\text{Dimension-Name}, \text{Rating}]\). Notice that both attributes are keys. As an example, \([\text{Accuracy}, 1]\) may be related to an attribute \(\text{cost}\) and states that such attribute has an accuracy equal to 1. This representation assumes that all attributes are rated on the same scale. An alternative representation to fix this point is \([\text{Dimension-Name}, \text{Attribute}, \text{Rating}]\).

- a Data Quality Measure entity, as \([\text{Rating}, \text{description}]\). For instance, \([1, \text{excellent}]\) for accuracy. Analogous to above, representation taking into account different scales is \([\text{Dimension-Name}, \text{Attribute}, \text{Rating}, \text{description}]\).

In the ER model, an attribute can’t be directly associated to another entity, such as a Data Quality Dimension entity or a Data Quality Measure entity. Therefore, an Attribute-Gerund-Representation is introduced, in which a gerund entity is introduced as a placeholder to solve this problem.

In this thesis, we define a model to associate quality data to data; this model is called \textit{Data and Data Quality} (\(D^2Q\) model and will be described in Section 3.2. Motivations according to which we chose to introduce a new model, instead of staying with the QER model are mainly related to the semistructured nature of the \(D^2Q\) model, and specifically:

- the semistructured nature allows to associate quality data to data in a more flexible way, with respect to structured data models, like QER;

- a semistructured data model can be easily implemented by using XML related standards. Especially in cooperative context in which the interoperability issues are particularly important, the usage of such standards becomes very relevant;

- the semistructured nature of the model allows to capture also semistructured data sources, such as XML data bases, that can be present in CIS’s in addition to structured relational data bases.
2.2.2 Extending the Relational Model: The Polygen Model [97] and the Attribute-based Model [99]

The Polygen Model addresses the problem of identifying, in the context of federated databases, which are the sources of data and which are the intermediate sources used to derive data. The Polygen Model extends the relational model with tags identifying local databases from which data are originated and intermediate local databases.

The Attribute-based data model tags data with quality indicators, which are characteristics of data. More specifically, an attribute may have an arbitrary number of quality indicators; an attribute in a relational scheme is expanded into an ordered pair called *quality attribute*, consisting of `<attribute, quality key>`. The expanded scheme is the *quality scheme*. The corresponding cell in the relational tuple is expanded into an ordered pair called *quality cell*, consisting of `<attributevalue, quality key value>`. The resulting relation is the *quality relation*. Each quality key value refers to a quality indicator tuple. In the Figure 2.6, an example of the model is shown.

The cell-tagging level of the model is also obtainable with the $D^2Q$ model, in which quality values can be associated to the leaves of a data graph (see 3.2). The main difference of the $D^2Q$ model is that we can associate quality values to different levels of the data structure, by exploiting characteristics of a semistructured data model.
2.3 Measuring & Improving Data Quality

A recent survey on methods for evaluating and improving data quality is [106]; in this paper two classes of methods are surveyed, namely: edit/imputation methods and record linkage methods.

Edit/imputation methods verify that data values satisfy pre-determined business rules. Fellegi and Holt [40] defined a formal mathematical model for data editing that is intended to minimize changes in data records and to assure that the substituted data values pass the edit rules. The means of implementing the model of Fellegi and Holt have primarily involved operations research [105].

Record Linkage methods have the task of identifying if two records are related to the same real world entity.

An automatic record linkage process generally consists of three different phases:

1 Pre-processing: it consists of parsing data and replacing different spellings or abbreviations with a standardized format. As an example, different spellings and abbreviations for the word “database” can be replaced by the single standardized spelling “DB”.

2 Comparison for matching: in order to decide for matching or not two records, it is necessary to compare them by using exact or approximate string comparators.

3 Decision for matching: an algorithm decides if declaring match, non-match or possible match for two compared records.

In the following of this section, we will focus on record linkage methods, not considering edit/imputation methods. Indeed, in the DaQuinCIS framework we had to solve the problem of linking together data from different sources, as described in Chapter 3.

2.3.1 Record Linkage Methods

Record Linkage is the task of linking together data representing the same real world entity and potentially coming from different sources. Record Linkage is also known as Record Matching, or the Object Identity problem [98]. In this thesis, we use the terms record linkage and record matching indifferently.

Many algorithms for record matching have been proposed. A first trend is mainly based on theoretical research and is most related to statistical and probabilistic areas. Such a trend started in 1959 with Newcombe, who first proposed an automatic method for record matching [45]; it continued with the work of Fellegi and Sunter [41], who developed a mathematical model that
has been improved and extended in different works, among which [10, 104]. In such works, probabilistic measures related to record matching are proposed, while nothing is said about matching characteristics of single records.

A second trend has an empirical basis and is most related to computer science area. Differently from the first trend, record matching measures are deterministic and matching features of specific records are fully determined. Among the first works, [98] provides a general description of the record matching problem in integrating different systems. In [48], 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. A method of detecting duplicate records within a data source without any distinguishing key data elements is the k-way sorting method [39].

Sorted-Neighborhood Method

The basic “sorted-neighborhood method” was proposed in [48, 47]. Given a collection of two or more databases, first a sequential list of records from databases and the sorted-neighborhood method is applied.

Algorithm. The sorted-neighborhood method can be summarized in three phases:

CREATE KEYS Compute a key for each record in the list by extracting relevant fields or portions of fields. The effectiveness of the sorted neighborhood method highly depends on a properly chosen key with the intent that common but erroneous data will have closely matching keys.

SORT DATA Sort the records in the data list using the key of step 1.

MERGE Move a fixed size window through the sequential list of records limiting the comparisons for matching records to those records in the window. If the size of the window is \( w \) records, then every new record entering the window is compared with the previous \( w-1 \) records to find “matching” records. (See Figure 2.7).

Selection of Keys. The effectiveness of the sorted-neighborhood method highly depends on the key selected to sort the records. A key is defined to be a sequence of a subset of attributes, or substrings within the attributes, chosen from the record.

As an example, let us consider the four records displayed in Figure 2.8. For this particular application, let us suppose that the “key designer” for
the sorting phase has identified a key as follows. The key consists of the concatenation of several ordered fields (or attributes) in the data: the first three consonants of a last name are concatenated with the first three letters of the first name field, followed by the address number field, and all of the consonants of the street name. This is followed by the first three digits of the social security field.

These choices are made since the key designer supposes that last names are typically misspelled (due to mistakes in vocalized sounds, vowels), but first names are typically more common and less prone to being misunderstood and hence less likely to be recorded incorrectly.

In the DaQuinCIS framework, we have extended the sorted-neighborhood method by making automatic the choice of the key to perform the matching. Instead of relying on "key designers" our idea is to rely on quality data associated to exported data values, in order to choose a "good" key for matching. In CIS's the key design problem is very complex, because of the multiple and different data sources, therefore an automatic support is necessary. The method is also applicable in any case in which quality metadata are available on data to be matched. The method is implemented by a module of the DaQuinCIS framework called Record Matcher and is used within a strategy for quality improvement that consists of: (i) linking different copies of the same data available in the CIS; (ii) improving on the base of the best quality copy. The Record Matcher is described in Section 4.1.
Multi-Pass Approach. In general, no single key will be sufficient to catch all matching records. The attributes or fields that appear first in the key have higher discriminating power than those appearing after them. Hence, if the error in a record occurs in the particular field or portion of the field that is the most important part of the key, there may be little chance a record will end up close to a matching record after sorting.

For instance, if an employee has two records in the database, one with social security number 193456782 and another with social security number 913456782 (the first two numbers were transposed), and if the social security number is used as the principal field of the key, then it is very unlikely both records will fall under the same window, i.e. the two records with transposed social security numbers will be far apart in the sorted list and hence they may not be merged.

To increase the number of similar records merged, the strategy implemented by [47] is to execute several independent runs of the sorted neighborhood method, each time using a different key and a relatively small window. They call this strategy the multi pass approach.

2.3.2 Process-based Improvement

In [80] two categories of methods for improving data quality are identified, namely: data-based methods and process-based methods.

Beyond a comparison of data with the real-world counterpart, data-based methods includes the two cited category of edit/imputation methods and record linkage methods, described in the previous section.

The principal problem of quality improvement by data-based methods is that those methods don’t prevent from future errors. In order to definitively solve the problem of poor quality, it is necessary to identify root causes of errors and redesign processes accordingly. Process-based methods have the major advantage of solving data quality problems on the long term, though they may be by far more expensive than data-based methods.

In the following of this section we will briefly overview the main proposals that enables process-based data quality improvement. In the DaQuinCIS
Functions of Information Processing (see Chapter 8 of [80]).

A process is generally composed by many activities; nevertheless, in order to point out data quality problems, it is important representing information flows involved in a process. Functions of Information Processing (FIP) is a model able to represent such information flows by means of an information model. The information model considers how data are created, moved, stored and so forth.

A manufacturing process is always composed by an input procedure where raw materials are acquired, a transformation procedure where the main activity of the process is executed and an output procedure where the final products are distributed. An information process operates in a similar way; raw materials are input data that are transformed in a structured process and the output data are information stored in a database. FIP focuses on the description of dynamic aspects of a process in terms of data. The dynamic functions that are used are: Associate, Filter, Prompt, Queue, Regulate, Store and Transmit. The descriptions of data manipulations are of the form:

\[
\text{Data Set A FIP Data Set B} = \text{Data Set C}
\]

The data sets on the left-hand side are called Input Information Products and those on the right are called Output Information products. It is intuitive that the quality of the output depends on the quality of the data in input and of the functions used.

Quality Function Deployment [5].

The Quality Function Deployment (QFD) is a tool for translating user requirements into objective technical specifications. The model is composed by matrices, whose elements describe the relationships among user requirements and technical features. QFD allows to analyze the information chain translating the customer requirements in specific technical requirements for data quality and performance specifications for the process that manipulates data. The method consists of five steps:

- Understand what customers want in their terms;
- Develop a single set of consistent user requirements;
- Translate user requirements into data quality requirements;
- Map each data quality requirements into one or more performance requirements;
Establish individual process performance and supplier requirements.

This method can be applied in design process to establish technical requirements based on data quality requirements.

**Ballou’s Information Manufacturing System** [8].

Ballou suggested to model an Information Manufacturing System in terms of specific blocks: processing block, data storage block, quality block, customer block and data vendor block. Starting from this specification, a tracking of timeliness, data quality and costs is carried on. The results are summarized in the so-called Information Manufacturing Analysis Matrix the values of which are compared with customers’ requirements in order to re-engineer the system to improve quality parameters.

**IP-MAP framework** [88]

An Information Production Map (IP-MAP) is a graphical model designed to help people to comprehend, evaluate, and describe how an information product such as an invoice, customer order, or prescription is assembled. It complements and extends Ballou’s Information Manufacturing System. In Chapter 5, we provide a detailed description of IP-MAP’s; indeed we have based the DaQuinCIS methodology on such an approach. The choice of IP-MAP’s as a basis for our methodology is mainly motivated by the following features:

- It effectively combines data and process representation. Differently from FIP, data are not simply input and output of process activities, but they have their own structure.
- Differently from QFD, the IP-MAP framework is focused on designing improvement actions a-posteriori, rather than taking care of quality requirement satisfaction in a system that is going to be realized.

### 2.4 Cooperative Information Systems

A Cooperative Information System (CIS) consists of a large number of cooperating systems, distributed over large and complex computer and communication networks, which work together cooperatively, requesting and sharing information, constraints, and goals [20, 67].

An information system which is part of a CIS shares goals with other agents in its environment, such as other information systems, human agents and the organization itself, and contributes positively towards the fulfillment
of these common goals. Cooperation with other information systems requires the ability to exchange information and to make a system’s own functionality available to other systems: this is often referred to as interoperability.

However, cooperation is more complex than simple sharing and interoperability; cooperation is the basic mean through which groups and organizations, while performing for their customers, continuously redesign themselves and their business processes, modify their boundaries, tune their objectives, and open themselves to new possibilities.

Supporting cooperation requires the system to be capable of reflecting both the changes that are decided for its performances (e.g., introducing new technologies) and the continuously ongoing changes of the organizational practices. The problem is not how to build information systems which share goals with their organizational environment, human users, and other existing systems at some time point. Rather, the problem is how to build information systems which continue to share goals with their organizational environment, human users, and other existing systems as they all evolve. It is the continuous organizational and technological change that makes CIS’s a challenge [28]: a CIS is not simply a collection of databases, applications and interfaces, rather it is an architectural framework which maintains consistency among a variety of computer-based systems, user groups, and organizational objectives as they all evolve over time.

CIS’s are a relatively young research area whose birth in the early 1990’s has been marked by the launching of an international journal, an ongoing conference, an international foundation and dedicated special issues in international journals. Different approaches have been proposed in the design and development of CIS’s; as pointed out in [20, 67], research on CIS’s addresses four basic areas:

- **Interoperation**: this includes topics such as open architectures; distributed object management; agent-based or network-centric computing; component-based applications; factoring out global control from individual components; communication protocols; translation mechanisms; data integration mechanisms; semantic meta-data repositories; knowledge sharing; blackboard architectures.

- **Cooperation**: this includes computer-supported collaborative work; synchronous and asynchronous sharing; virtual workspaces; concurrency control; multiagent systems; transaction management; mediation architectures; business process and workflow management systems; artificial intelligence planning; multiagent technologies; intelligent scheduling; self-describing systems; reflective architectures.

- **Information Services**: these include cooperative querying; information
retrieval; data mining; meta-data management; data warehouses; information brokering; knowledge sharing; knowledge-level communication protocols; heterogeneous and distributed databases; information access on the Web.

Organizational Studies and Change Management: this includes changes dictated by technology and/or organizational objectives, and covers topics such as constraint enforcement; schema evolution; database view updates; artificial intelligence theories of action; truth maintenance systems; constraint satisfaction; versions and configurations; impact analysis; risk assessment; business process reengineering; enterprise integration.

With respect to these four basic areas, the work presented in this thesis mainly concerns the Interoperation and Information Services areas; specifically, the thesis proposes a data model to associate quality to data and a suite of services for querying and improving the quality of data exchanged by cooperating organizations.

2.5 Data Integration Systems

The main component of the DaQuinCIS framework, namely the Data Quality Broker described in Chapter 3, is a data integration system that allows to query for retrieving data with a specified quality available at sources.

This section describes how data quality related issues are considered in some data integration systems. Analogies and differences with the Data Quality Broker are also described in detail, by focusing both on aspects directly connected to the theoretical issues involved in the Data Quality Broker design, and on aspects that are instead strictly but not directly associated to the Data Quality Broker design.

Specifically, after a brief overview of the characteristics of a data integration framework in Section 2.5.1, we review some proposals that take data quality into account as the principal objective of the research 2.5.2. Then, in Section 2.5.4, we describe some systems that does not directly consider data quality, but they focus on issues that are inherently related to it.

2.5.1 Basics on Data Integration Systems

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of this data, called global schema. A Data Integration System is composed by three elements: (i) a global schema; (ii) a source schema, including schemas of all sources and (iii) a mapping between the global schema and the source schema.
Two basic approaches have been proposed to specify the mapping. The first approach, called Global-as-View (GAV) requires that the global schema is expressed in terms of the data sources. The second approach, called Local-as-View (LAV) requires that each data source is expressed as a view over the global schema.

A data integration system [53] can be formally defined as a triple \((G, S, M)\) where:

- \(G\) is the global schema, expressed in a language \(L_G\) over an alphabet \(A_G\).
- \(S\) is the source schema, expressed in a language \(L_S\) over an alphabet \(A_S\).
- \(M\) is the mapping between \(G\) and \(S\), constituted by a set of assertion of the forms: \(q_s \leadsto q_g\) and \(q_g \leadsto q_s\), where \(q_g\) and \(q_s\) are two queries of the same arity respectively over the global schema \(G\) and the source schema \(S\). Queries \(q_S\) are expressed in a query language \(L_{M,S}\) over the alphabet \(A_S\), and queries \(q_G\) are expressed in a query language \(L_{M,G}\) over the alphabet \(A_G\).

Given a data integration system \(I = (G, S, M)\), we now want to assign a semantics to it. Let \(D\) be a source database for \(I\), i.e. a database that conforms to the source schema \(S\) and satisfies all constraints in \(S\). Based on \(D\) we now specify which is the the information content of the global schema \(G\). We call global database for \(I\) any database for \(G\). A global database \(B\) is said to be legal with respect to \(D\), if:

- \(B\) is legal with respect to \(G\), i.e. \(B\) satisfies all the constraints of \(G\);
- \(B\) satisfies the mapping \(M\) with respect to \(D\).

The notion \(B\) satisfies the mapping \(M\) with respect to \(D\) depends on how to interpret the assertion and is discussed in the next section.

**Query processing**

Irrespectively if the mapping is GAV or LAV, query processing in data integration requires a reformulation step: the query posed over the global schema has to be reformulated in terms of a set of queries over the sources. Nevertheless, the actual realization of query processing in a data integration systems is strictly dependant from the method used for the specification of the mapping.

Query processing in GAV can be based on a simple unfolding strategy, if there are not integrity constraints on the global schema. Given a query \(q\) over the alphabet of the global schema \(A_G\), every element of \(A_G\) is substituted with the corresponding query over the sources, and the resulting query is then evaluate data the sources.
Since in LAV, sources are modeled as views over the global schema, the
problem of processing a query is called view-based query processing. There
are two approaches to view-based query processing, namely: view-based query
rewriting and view-based query answering. View-based query rewriting con-
sists of reformulating the query into a possibly equivalent expression, called
rewriting, that refers only to the source structures. Once the rewriting of
the query has been computed, it can be directly evaluated over the source to
obtain the answer to the query. View-based query answering is more direct:
besides the query and the mapping definitions, we are also given the exten-
sions of the views over the global schema. The goal is to compute the set of
tuples that are the answer set of the query in all databases that are consistent
with the information on the views.

As described in Chapter 3, the Data Quality Broker of the DaQuinCIS
system adopts a GAV approach, therefore, an unfolding strategy is used for
query processing. Notice that, the unfolding strategy is not straightforward, as
it is complicated by the semistructured nature of the global and local schemas.

2.5.2 Data Quality Driven Integration Systems

Data quality has been explicitly addressed in few works. In this section we
provide a short description of three works, namely: Naumann’s planning al-
gorithm, described in [70], the MIT Context Interchange project [19], the
proposal for quality-driven querying in [64]. We also compare these works
with DaQuinCIS , outlining the major differences.

Quality-driven planning [70]

In [70], an algorithm for querying for best quality data in a LAV integration
system is proposed. The mapping between the local schemas and the global
schema are expressed by means of assertions called Query Correspondence
Assertions (QCA’s).

Quality values are statically associated to QCA’s and to data sources.
Instead, some quality values are associated to user queries at query time.

The planning algorithm allows to select the best quality plan to answer a
query in three steps:

- Source selection, that cuts off poor quality sources before starting the
  actual planning; the aim is reducing the number of generated plans.
- Plan creation, that generates the set of plans answering a query accord-
  ing to a classical LAV rewriting.
- Plan selection, that orders plans on the basis of quality values and selects
  the best quality ones.
In the DaQuinCIS framework, we share with Naumann’s work the idea of querying for best quality data.

But differently from his work, we focus on:

- a model according to which export data and quality data;
- querying such a model in a GAV data integration system.

The proposal of a model that associates data to quality data is motivated by many issues. First of all, having a model allows us to associate quality values at different granularity levels, i.e. we can associate a quality value to each data value and object. This is an actual requirement of real systems, for which it is not enough to know that “medium accuracy” of a field is good: it is wanted that the requested record has a good accuracy, just the requested one. Secondly, modeling together data and quality data provides a useful abstraction level for users, who have not to solve separately the problem of retrieving data and quality data; quality data may be asked by using the available querying service.

Moreover, we have chosen a GAV data integration system, instead of a LAV one, as our setting does not consist of web sources: we propose DaQuinCIS for a cooperative context, in which the number of organizations is known and slowly variable in time. But the main difference of our work with respect to Naumann’s one is the semantics of our system. Our aim is not only querying, but also improving quality of data. To such a scope, the query processing step has a specific semantics that allows to send improvement feedbacks on each query (see Chapter 3).

The MIT Context Interchange Project [19]

The MIT Context Interchange project is based on the idea of modeling a “context” for integrating heterogeneous sources. Such a context consists of metadata that allows for solving problems such instance level conflicts that may occur in the data integration phase. An integration system based on the LAV approach implements query processing with contexts.

This approach takes into account a specific dimension of data quality, namely data interpretability. Indeed, a context allows for clarifying that a price is, for example, in Dollar or in Yen.

The approach adopted in DaQuinCIS is more general as far as quality dimensions considered. DaQuinCIS allows for querying and improving data quality in general, while COIN is a system thought to resolve problems related to a specific quality dimension.
2.5.3 Quality-driven source selection [64].

In [64], the basic idea is querying web data sources by selecting them on the basis of quality values on provided data. Specifically, the authors suggest to publish metadata characterizing the quality of data at the sources. Such metadata are used for ranking sources, and a language to select sources is also proposed.

In the DaQuinCIS system, we associate quality to data (at different granularity levels) rather than to a source as a whole. This makes things more difficult, but allows to pose more specific queries. For instance, the DaQuinCIS system easily treats the quite common cases in which a source has some data which have a low quality and some other ones that have instead a higher quality, by making the source be an actual data provider only for better quality data.

2.5.4 Data Quality Issues in Data Integration Systems

In a data integration system, a query posed over the global schema is reformulated in terms of a set of queries over the sources. Once such queries are executed locally, results need to be reconciled and merged. This is an essential phase of a data integration process and is called data reconciliation phase [18]. The data reconciliation phase deals with problems at instance level, whereas data conflicts have to be resolved. As also recognized in [83], while several methods for schema integration have been proposed in the past, the problem of instance integration is addressed only partially. In [79] an interesting classification of data quality problems is provided. This classification distinguishes: (i) Single-Source problems (at schema and at instance level), and (ii) Multi-source problems (at schema and at instance level). Examples of single-source problems at instance level are misspelling, duplicates, incompleteness etc. Examples of multi-source problems at instance level are mainly provided by inconsistent data among the different sources.

Data integration systems dealing with instance level conflicts only focus on multi-source problems at instance level. Examples of such systems are the interactive system proposed in [83] and the AURORA system proposed in [107]. The system described in [83] 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. AURORA supports conflict tolerant queries, i.e. it provides a dynamic mechanism to resolve conflicts by means of defined conflict resolution functions.

In the DaQuinCIS system, we introduce a set of quality values associated to data at each source; then we can rely on such values when performing a
comparison of different copies of data, thus having more chances of solving multi-source conflicts.
Chapter 3

The DaQuinCIS Framework: the $D^2Q$ model and the Data Quality Broker

3.1 Overview: The DaQuinCIS Framework

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 [9].

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 thesis, 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 $N$, which provide software services to organizations as well as reliable con-
Each organization \( \text{Org}_i \) is connected to \( \mathbb{N} \) through a gateway \( G_i \), on which software services offered by \( \text{Org}_i \) 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.

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 this chapter, we describe the DaQuinCIS architecture that allows 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 on the specific activities to be carried out. The overall architecture is depicted in Figure 3.1.

Organizations export data and quality data according to a common model, referred to as \textit{Data and Data Quality (D²Q) 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 3.2.

In order to produce data and quality data according to the \( D^2Q \) model, each organization deploys on its cooperative gateway a \textit{Quality Factory} service that is responsible for evaluating the quality of its own data. In Section 4.3, we provide an overview of the main functionalities of the Quality Factory.

The \textit{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 \textit{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 \textit{quality improvement function}. 

In order to produce data and quality data according to the \( D^2Q \) model, each organization deploys on its cooperative gateway a \textit{Quality Factory} service that is responsible for evaluating the quality of its own data. In Section 4.3, we provide an overview of the main functionalities of the Quality Factory.

The \textit{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 \textit{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 \textit{quality improvement function}. 

The DaQuinCIS framework: The D²Q model and the Data Quality Broker

Connectivity. Each organization \( \text{Org}_i \) is connected to \( \mathbb{N} \) through a gateway \( G_i \), on which software services offered by \( \text{Org}_i \) to other organizations are deployed. A user is a software or human entity residing within an organization and using the cooperative system.
3.1. OVERVIEW: THE DAQUINCIS FRAMEWORK

The Data Quality Broker is in essence a peer-to-peer data integration system 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 3.4 and in Section 3.5.

The Quality Notification Service is a publish/subscribe engine used as a quality message bus between services and/or organizations. More specifically, it allows quality-based subscriptions for users to be notified on changes of the quality of data. For example, an organization may want to be notified if the quality of some data it uses degrades below a certain threshold, or when high quality data are available. Also the Quality Notification Service is deployed as a peer-to-peer system. An overview of the Quality Notification Service is provided in Section 4.3.

The Rating Service associates trust values to each data source in the CIS. These values are used to determine how much an organization can be trusted with respect to provided data. The design of the Rating Service is discussed in detail in Section 4.2.

The remaining of this chapter is organized as follows. After the description of a running example, in Section 3.2 we describe the \( D^2Q \) model. Section 3.3 explains how the \( D^2Q \) model can be queried by XQuery. In Section 3.4, the overall design of the Data Quality Broker is presented, while in Section 3.5 details are provided on how the mediation function of the Data Quality Broker has been realized.
3.1.1 Running Example

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 [4] 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 Accident 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 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 3.1.1, three simplified versions of the local schemas are shown.

3.2 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

\footnote{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.}
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 (D²Q) model. In this section, we first introduce the data quality dimensions used in the model (Section 3.2.1), then we describe the D²Q model with respect to the data features (Section 3.2.2) and the quality features (Section 3.2.3).

### 3.2.1 Data Quality Dimensions

In this section, we define the set of data quality dimensions used in this work. The need for providing such definitions stems from the lack of a common reference set of dimensions in the data quality literature. We propose four dimensions for data quality; such dimensions are shared by different quality dimensions proposals, as discussed in Section 2.1.1. Actually, we do not consider two dimensions that have the same characteristic of being present in many proposals, namely interpretability and accessibility. The reason is that we have chosen to focus on more objective quality dimensions; nevertheless, the proposed set is simply a basic reference one and does not want to be exhaustive for a data quality characterization.

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) consistency.

**Accuracy**

In [80], 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 [46]; such functions consider the minimum number of operations on individual characters needed in order to transform a string into another.

**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 [78] 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 3.2.2, 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).

**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 \[8\]; 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” \[65\] or a “transaction time” like in temporal databases \[91\].

**Consistency**

Consistency implies that two or more values do not conflict each other. We refer to an internal type of consistency, meaning 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, consistency can be defined as:

*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 \(v = \text{JOHN}\) 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.
CHAPTER 3. THE DAQUINCIS FRAMEWORK: THE D²Q MODEL
AND THE DATA QUALITY BROKER

Consistency has been widely investigated both in the database area through integrity constraints (e.g., [25]) and in the statistics area, through edits checking (e.g., [21]).

3.2.2 The D²Q Data Model

The D²Q model is a semistructured model that enhances the semantics of the XML data model [42] in order to represent quality data. In the following of this section we provide a formal definition for a data class and a D²Q data schema. D²Q data schemas define the data portion of the D²Q model; instead, in Section 3.2.3, we define D²Q quality schemas, corresponding to the quality portion of the D²Q model.

Definition 3.2.1 [Data Class] A data class $\delta$ (name$_\delta$, $\pi_1, \ldots, \pi_n$) consists of:
- a name name$_\delta$;
- 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 $\pi_i$ and type$_i$ can be:
  - either a basic type$^2$;
  - or a data class;
  - or a type set-of < X >, where < X > can be either a basic type or a data class.

□

Definition 3.2.2 [D²Q Data Schema] A D²Q data schema $\mathcal{S}_D$ is a direct, node- and edge-labelled graph with the following features:
- a set of nodes $\mathcal{N}_D$, each of which is a data class;
- a set of leaves $\mathcal{T}_D$, that are all basic type properties;
- a set of edges $\mathcal{E}_D \subseteq \mathcal{N}_D \times (\mathcal{N}_D \cup \mathcal{T}_D)$;
- a set of nodes and leaves labels $\mathcal{L}_D = \mathcal{L}_{\mathcal{N}_D} \cup \mathcal{L}_{\mathcal{T}_D}$, where $\mathcal{L}_{\mathcal{N}_D}$ is the set of node labels, each node label consisting of < data class name: data class >; $\mathcal{L}_{\mathcal{T}_D}$ is the set of leaves labels, each leaf label consisting of < property name: basic type >.

$^2$Basic types are the ones provided by the most common programming languages and SQL, that is Integer, Real, Boolean, String, Date, Time, Interval, Currency, Any.
3.2. THE $D^2Q$ MODEL

Figure 3.3: A $D^2Q$ data schema of the running example

- a set of edge labels $L_{E_D}$ defined as follows. Let $< n_1, n_2 > \in E_D$. Let $n_1$ be a data class and let $l_{1,2}$ be the label on the edge $< n_1, n_2 >$. Such a label is $l_{1,2} = \text{number of occurrences of } n_2$, i.e. the label specifies the cardinality of $n_2$ in the relationship with $n_1$. $^3$

□

A $D^2Q$ data schema must have the two following properties.

Closure Property. If $S_D$ contains a data class $\delta$, having as a property another data class $\delta'$, then also $\delta'$ belongs to $S_D$.

Aciclicity. No cycle exists in a $D^2Q$ schema.

In Figure 3.3, the $D^2Q$ data model of the INPS local schema of the running example is shown. Specifically, two data classes, enterprise and owner, are represented. The edge connecting enterprise to owner is labelled with $*$, indicating the n-ary cardinality of owner; the edges connecting each data class with the related basic type properties are labelled with 1 indicating the cardinality of such properties.

3.2.3 THE $D^2Q$ Quality Model

This section describes the model we use to represent quality data, i.e. the $D^2Q$ quality model, for which we define the concepts of quality class and quality schema.

$^3$We assume that it is possible to specify the following values for minimum and maximum cardinality of relationships: 1..1, 0..1, 1..* (where * stands for n-ary relationship), 0..*, x..y (where x and y are any fixed integer different from 0 and 1), 0..y, 1..y, 1..*. If no minimum cardinality is specified it is assumed the minimum cardinality is 1.
Besides basic data types, we define a set of types that are called quality types. Quality types correspond to the set of values that quality dimensions can have. We indicate as:

- $\tau_{\text{accuracy}}$, $\tau_{\text{completeness}}$, $\tau_{\text{consistency}}$, $\tau_{\text{currency}}$,

the quality types considered in our model.

More specifically, each quality type is defined over a domain of values, referred to as $\text{Dom}_{\text{accuracy}}$, $\text{Dom}_{\text{completeness}}$, $\text{Dom}_{\text{consistency}}$, and $\text{Dom}_{\text{currency}}$. Such domains are defined according to the metrics used to measure quality values; moreover each domain is enhanced with the special symbol $\bot$ to indicate that no quality value is defined for that dimension. For instance, if we adopt a 3-value scale to measure accuracy, then $\text{Dom}_{\text{accuracy}} = \{\text{High, Medium, Low, } \bot\}$.

A quality class is associated to each data class and to each property. We use the following notation: if $\delta (\text{name}_\delta, \pi_1, \ldots, \pi_i, \ldots, \pi_n)$ is a data class, then $\lambda_\delta$ is the associated quality class and $\lambda_{\delta;\pi_i}$ is the quality class associated to $\pi_i$. In the following, we will first define $\lambda_{\delta;\pi_i}$, where $\pi_i$ is a basic type property; then we define the quality class $\lambda_\delta$.

Definition 3.2.3 $[\lambda_{\delta;\pi_i}-\text{Quality class associated to a (basic type) property of a data class}]$ Let $\delta (\text{name}_\delta, \pi_1, \ldots, \pi_i, \ldots, \pi_n)$ be a data class. Let $\pi_i$ be a basic type property of $\delta$.

The quality class $\lambda_{\delta;\pi_i}$ is defined by:

- a name $\text{name}_{\lambda_{\delta;\pi_i}}$;
- four properties $\kappa_{\text{accuracy}}$, $\kappa_{\text{completeness}}$, $\kappa_{\text{consistency}}$, $\kappa_{\text{currency}}$, such that
  
  $$(\kappa_{\text{accuracy}} : \tau_{\text{accuracy}}) \cup (\kappa_{\text{completeness}} : \tau_{\text{completeness}}) \cup (\kappa_{\text{consistency}} : \tau_{\text{consistency}}) \cup (\kappa_{\text{currency}} : \tau_{\text{currency}}).$$

Definition 3.2.4 $[\lambda_{\delta}-\text{Quality class associated to a data class}]$ Let $\delta (\text{name}_\delta, \pi_1, \ldots, \pi_n)$ be a data class. A quality class $\lambda_\delta (\text{name}_\lambda_\delta, \kappa_{\text{accuracy}}, \kappa_{\text{completeness}}, \kappa_{\text{consistency}}, \kappa_{\text{currency}}, \pi_1^\lambda, \ldots, \pi_n^\lambda)$ consists of:

- a name $\text{name}_\lambda_\delta$;
- a set of properties $(\kappa_{\text{accuracy}}, \kappa_{\text{completeness}}, \kappa_{\text{consistency}}, \kappa_{\text{currency}}, \pi_1^\lambda, \ldots, \pi_n^\lambda)$ such that:
  
  - $\kappa_{\text{accuracy}}$, $\kappa_{\text{completeness}}$, $\kappa_{\text{consistency}}$, $\kappa_{\text{currency}}$ are
  
  such that
  
  $$(\kappa_{\text{accuracy}} : \tau_{\text{accuracy}}) \cup (\kappa_{\text{completeness}} : \tau_{\text{completeness}}) \cup (\kappa_{\text{consistency}} : \tau_{\text{consistency}}) \cup (\kappa_{\text{currency}} : \tau_{\text{currency}}).$$
3.2. THE $D^2Q$ MODEL

- $(\pi_1^\lambda, \ldots, \pi_n^\lambda)$ is a set of properties $\pi_i^\lambda = \langle \text{name}_i^\lambda : \text{type}_i^\lambda \rangle$, that biunivocally corresponds to $(\pi_1, \ldots, \pi_n)$ (properties of $\delta$) and is defined as:
  * if $\pi_i$ is a basic type property, then $\text{type}_i^\lambda$ is $\lambda_{\delta::\pi_i}$, where $\lambda_{\delta::\pi_i}$ is the quality class associated to $\delta :: \pi_i$;
  * if $\pi_i$ is a set of $< X >$ type where $X$ is a basic type, then $\text{type}_i^\lambda$ is set of $< \lambda_{\delta::\pi_i} >$, where $\lambda_{\delta::\pi_i}$ is the quality class associated to $\delta :: \pi_i$;
  * if $\pi_i$ is a data class $\delta'$, then $\text{type}_i^\lambda$ is $\lambda_{\delta'}$, where $\lambda_{\delta'}$ is the quality class associated to $\delta'$;
  * if $\pi_i$ is a set of $< X >$ type where $X$ is a data class $\delta'$, then $\text{type}_i^\lambda$ is set of $< \lambda_{\delta'} >$, where $\lambda_{\delta'}$ is the quality class associated to $\delta'$.

□

Definition 3.2.5 [D$^2$Q Quality Schema] A $D^2Q$ quality schema $S_Q$ is a direct, node- and edge-labelled graph with the following features:

- a set of nodes $N_Q$, each of which is a quality class;
- a set of leaves $T_Q$, that are quality type properties;
- a set of edges $E_Q \subseteq N_Q \times (N_Q \cup T_Q)$;
- a set of node and leaves labels $L_Q = L_{N_Q} \cup T_Q$ where $L_{N_Q}$ is the set of node labels, each node label consisting of $< \text{quality class name: quality class }>$; $L_{T_Q}$ is the set of leaves labels, consisting of $< \text{quality property name: quality type }>$;
- a set of edge labels $L_{E_Q}$ defined as follows. Let $< n_1, n_2 > \in E_Q$. Let $n_1$ be a quality class and let $l_{1,2}$ be the label on the edge $< n_1, n_2 >$; such a label is defined in dependance on $n_2$ as follows:
  - if $n_2$ is a leaf, then $l_{1,2}$ is the NULL string, as we do not need to specify any cardinality;
  - if $n_2$ is a quality class, then then $l_{1,2}=$number of occurrences of $n_2$, i.e. the label specifies the cardinality of $n_2$ in the relationship with $n_1$.

□
Let us consider again the INPS local schema of the running example; as seen, the $D^2Q$ data schema is shown in Figure 3.3. In Figure 3.4, we instead represent the $D^2Q$ quality schema. Specifically, two quality classes, \texttt{Enterprise} and \texttt{Owner} are associated to the \texttt{Enterprise} and \texttt{Owner} data classes. Accuracy nodes are shown for both data classes and related properties. Note that also consistency, currency and completeness nodes are present in the graph, though not shown; this however does not prevent from the possibility of not specifying quality values, as the special symbol \texttt{⊥} has been added to the domains of data quality dimension types to such a scope.

### 3.2.4 The $D^2Q$ Model: Data + Quality

In the previous section, the $D^2Q$ quality schema has been defined according to a correspondence with the $D^2Q$ data schema. In this section, we formally define the relationship between $D^2Q$ data schema and the $D^2Q$ quality schema. Specifically, we define the notion of \textit{quality association} between nodes of the two graphs.

\begin{definition}[Quality Association] Let us call $A = \{N_D \cup T_D\}$ the set of all nodes of a $D^2Q$ data schema $S_D$; let us call $B = N_Q$ the set of all non-leaf nodes of a $D^2Q$ quality schema $S_Q$.

A \textit{quality association} is a function:

\[\text{qualityAss} : A \rightarrow B,\] 

such that:

\[\forall x \in A \exists a \ unique \ y \in B \ such \ that \ y = \text{qualityAss}(x) \land \ (\forall y \in B \exists \ a \ unique \ x \in A \ such \ that \ y = \text{qualityAss}(x)) \Rightarrow \text{qualityAss} \ is \ a \ biunivocal \ function.\]

\end{definition}
3.2. THE $D^2Q$ MODEL

It is easy to define, now, a $D^2Q$ schema that comprises both data and associated quality definitions.

**Definition 3.2.7 [D^2Q Schema]** Given the $D^2Q$ data schema $S_D$ and the $D^2Q$ quality schema $S_Q$, a $D^2Q$ Schema $S$ is a direct, node- and edge-labelled graph with the following features:

- a set of nodes $N = \{N_D \cup N_Q\}$, deriving from the union of nodes from $S_D$ and nodes from $S_Q$;
- a set of leaves $T = T_D \cup T_Q$;
- a set of edges $E = \{E_D \cup E_Q \cup E_{DQ}\}$, where $E_{DQ} \subseteq \{N_D \cup T_D\} \times N_Q$, specifically, special edges connecting nodes from $S_D$ to nodes from $S_Q$ are introduced whereas qualityAss is defined; such edges are added to the union of edges from $S_D$ and edges from $S_Q$;
- a set of node labels $L_N = L_D \cup L_Q$;
- a set of edge labels $L_E = L_{E_D} \cup L_{E_Q} \cup L_{E_{DQ}}$, where $L_{E_{DQ}} = \{quality\}$ is a set of quality labels.

The $D^2Q$ schema related to our example schemas in Figure 3.3 and in Figure 3.4 is shown in Figure 3.5 (for the sake of clearness, types are not shown).

$D^2Q$ Schema Instances

The instances of both $D^2Q$ data schemas and $D^2Q$ quality schemas can be easily derived according to the class-based structure of the $D^2Q$ model.

More specifically:

- data classes instances are *data objects*;
- similarly, quality classes instances are *quality objects*;
- quality association values corresponds to *quality links*.

A $D^2Q$ data schema instance is a graph with nodes corresponding to data objects and leaves corresponding to properties values. More specifically, nodes are labelled with data objects names and leaves are labelled with a couple
CHAPTER 3. THE DAQUINCIS FRAMEWORK: THE D²Q MODEL
AND THE DATA QUALITY BROKER

Figure 3.5: A $D^2Q$ schema of the running example

$<\text{basic type property name, basic type property value}>$. Edges are not labelled at all.

Similarly, a $D^2Q$ quality schema instance is a graph with quality objects nodes and leaves corresponding to quality properties values. Leaves are labelled by a couple $<\text{quality type property name, quality type property value}>$. Edges are not labelled at all.

Finally, a $D^2Q$ schema instance comprises the two previous ones plus edges actually connecting data objects and quality object, corresponding to quality links, labelled with a quality label.

Figure 3.6 shows the graph representation of a $D^2Q$ schema instance of the running example. The Enterprise1 object, instance of the Enterprise class, has two owners, i.e. Owner1 and Owner2, objects of the Owner class; the associated quality values are also shown (only quality links related to data objects are shown for the sake of simplicity).

3.2.5 Implementation of the $D^2Q$ Data Model

In this section, we describe how $D^2Q$ schemas are translated into XML Schemas [93, 16] that correspond to the local schemas to be queried in order to retrieve data with the associated quality in the CIS.

The results of queries posed over local schemas are $D^2Q$ schema instances that are exchanged as XML documents; the implementation of $D^2Q$ schema instances as XML documents is also described in the following of this section.

From a $D^2Q$ schema to an XML Schema

$D^2Q$ schemas represent in a formal and well-defined way: (i) the schema of data exported by each organization in the CIS, (ii) the schema of quality data
3.2. THE $D^2Q$ MODEL

Figure 3.6: $D^2Q$ schema instance of the running example

and (iii) the associations relating data to their quality. From an implementation point of view, we have chosen to export $D^2Q$ schemas as XML Schemas [93, 16].

Generally speaking, the XML Schema grammar allows one to define data types and to declare elements that use these data types; we exploit this property in order to represent a $D^2Q$ schema as an XML Schema. We chose to have two distinct documents representing respectively a $D^2Q$ data schema and the associated $D^2Q$ quality schema. This is motivated by making as flexible as possible the choice of exporting only data as local schemas or both data and quality. Indeed, for the end user it may be easier to use predefined functions that allow to access quality data, instead of directly querying them. This concept will be clarified later in this chapter, where we define quality functions to access quality data; however, it should be clear that having two separate schemas allows for both a direct and indirect query mechanism for quality data.

Starting from a $D^2Q$ schema, there are several ways to translate it into an XML Schema. We have chosen a translation that maps the graph structure of the $D^2Q$ schema into the tree structure of the data model underlying XQuery, which is the query language we chose to adopt. We could have chosen to directly represent a $D^2Q$ schema with a graph structure; indeed, XML allows to
have a graph data model by using mechanisms such as XLink [31] or XPointer [30]. Our idea was instead to have a query mechanisms as direct as possible, and in this sense the tree structure of XQuery was the best candidate.

The translation of the graph structure into a tree structure is implemented by replicating objects whereas necessary. Indeed, in the $D^2Q$ data model a data class can be a property of different data classes, i.e. it can have different parents. In these cases, the data class is replicated and at object level the same instances can be identified by explicitly introducing Object Identifiers (OIDs).

Quality links are implemented by Quality Object Identifiers (QOID’s) both on data and quality objects.

The principal choices for the representation of $D^2Q$ schemas as XML Schemas were:

- representing data and quality classes and also their properties as XML elements;
- limiting the use of XML attributes only to represent control information, i.e., object identifier and quality object identifier together with information to represent quality association;
- definition of XML Schema types that correspond to basic and quality types defined in the $D^2Q$ model;
- definition of a root element for each $D^2Q$ data schema;
- definition of a root element for each $D^2Q$ quality schema.

An example of definition of a quality type of the $D^2Q$ model as an XML Schema type is shown in figure 3.7, where the quality type accuracyType is defined. Notice that quality types are defined as simple types that are a restriction of XML Schema simple types.

The two XML schemas, corresponding to the $D^2Q$ schema shown in Figure 3.5, are shown in Figure 3.8 for the data portion and in Figure 3.9 for the quality portion.

Figure 3.7: An example of definition of a quality type as an XML Schema type.
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://DaQuincis.project/D2Q"
    xmlns="http://DaQuincis.project/D2Q" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:include schemaLocation="xsd/gendef.xsd"/></xs:schema>
<xs:complexType name="Enterprise_Class">
    <xs:sequence>
        <xs:element name="Name" type="String"/>
        <xs:element name="Code" type="String"/>
        <xs:element name="Owned" type="Owned_Class"/>
    </xs:sequence>
    <xs:attribute name="OID" type="OIDType" use="required"/>
    <xs:attribute name="qOID" type="qOIDType" use="required"/>
</xs:complexType>
<xs:complexType name="Owned_Class">
    <xs:sequence>
        <xs:element name="Owner" type="Owner_Class"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Owner_Class">
    <xs:sequence>
        <xs:element name="FiscalCode" type="String"/>
        <xs:element name="Address" type="String"/>
        <xs:element name="Name" type="String"/>
    </xs:sequence>
    <xs:attribute name="OID" type="OIDType" use="required"/>
    <xs:attribute name="qOID" type="qOIDType" use="required"/>
</xs:complexType>
</xs:schema>

Figure 3.8: XML Schema of the $D^2Q$ data schema of the running example.
CHAPTER 3. THE DAQUINCIS FRAMEWORK: THE $D^2Q$ MODEL
AND THE DATA QUALITY BROKER

Figure 3.9: XML Schema of the $D^2Q$ quality schema of the running example.
3.3. QUERYING THE $D^2Q$ MODEL WITH XQUERY

From a $D^2Q$ schema instance to an XML file

The translation rules from a $D^2Q$ schema instance to an XML file reflects the rules given for the $D^2Q$ schema translation, described in the previous section. In synthesis, two XML documents correspond to a $D^2Q$ schema instance: one is an instance of the $D^2Q$ data schema and the other one is an instance of the $D^2Q$ quality schema. They are obtained from the respective $D^2Q$ schemas according to the following rules:

☐ a root element is defined to contain all data objects;
☐ a root element is defined to contain all quality objects;
☐ an object $O$ instance of a data class maps into an XML element, with an OID attribute identifying the object and a qOID attribute pointing to the quality;
☐ a property of a data class is mapped into an XML element;
☐ similarly, an object $O_q$ instance of a quality class , is mapped into an XML element with a qOID identifying it;
☐ a property of a quality class is also mapped into an XML element.

We show in Figure 3.10 a portion of the XML file corresponding to the $D^2Q$ data schema instance shown in Figure 3.8.

3.3 Querying the $D^2Q$ model with XQuery

The $D^2Q$ model has been thought in order to be queryable in an easy way and by a standard language. XQuery [17] is the language we chose to query data and quality data. In the following of this section, we first briefly describe the basic concepts of XQuery. Then, in Section 3.3.2, we define a set of quality functions to select quality values associated to data in the $D^2Q$ model and we show some examples of their usage.

3.3.1 The XML query language: XQuery

XQuery is the W3C query language for XML, and provides features for retrieving and restructuring XML information from diverse data sources. In the following of this section, we provide a brief summary of the key features of XQuery, then we shortly illustrate the data model underlying XQuery and the navigation and query principal constructs. For more details, the reader should refer to [17].
CHAPTER 3. THE DAQUINCI’S FRAMEWORK: THE D²Q MODEL
AND THE DATA QUALITY BROKER

XQuery is a functional, strongly-typed language which allows various kinds of expressions to be nested, and requires the operands of various expressions, operators and functions to conform to designated types. The basic building block of XQuery is the expression. The language specifies many different kinds of expressions, including primary expressions, path expressions, sequence expressions, arithmetic expressions, comparison expressions, logical expressions, constructor expressions, FLWOR expressions, conditional expressions, and quantified expressions. Of these, the two that are most relevant to our work, are path expressions and FLWOR expressions; these are described in the following sections.

Each XQuery expression is evaluated in an expression context, consisting of the static context and the dynamic context, which consists of all the information that can affect the result of the expression. The static context consists of default namespaces and collations; in-scope namespaces, schema definitions, variables, functions and collations; and the base URI. The dynamic context consists of the input sequence of nodes processed by the expression; the dynamic types and values of the in-scope variables, provided by the execution of the XQuery expressions in which the variables are bound; and the expression focus, which enables the XQuery processor to keep track of which nodes are being processed by the expression. The focus consists of (i) the context item (node or atomic value), from the sequence of items, currently being processed, (ii) the context position, which is the position of the context item in this sequence of items, and (iii) the context size, which is the number of items in this

Figure 3.10: A portion of the XML file corresponding to data shown in the D²Q schema instance of the running example.
sequence of items. The value of an XQuery expression is always a sequence of zero or more items, i.e., a value in the XML data model.

XQuery Data Model

The XQuery 1.0 and XPath 2.0 data model [42] (referred to as the XML data model) serves to define the inputs to, and permissible results of, XQuery, XPath and XSLT languages. Every XML data model value is a sequence of zero or more items, where an item is either a node or an atomic value:

- A node may be of seven distinct kinds: document, element, attribute, text, namespace, processing instruction, and comment, structured as an ordered, node-labeled tree with node identity.

- An atomic value encapsulates an XML Schema atomic type [16], and a corresponding value of that type.

Each kind of node is characterized by two sets of abstract functions, which serve an explanatory, non-normative, role: (i) a constructor, which creates a new node with a unique identity, different from all other nodes, and (ii) a set of accessors, which are used to expose properties of the items in the data model. Examples of accessors include: $dm$::node-kind, which returns a string value identifying the kind of node on which the accessor is called; $dm$::parent, which returns the parent node of nodes that have a parent (and the empty sequence, otherwise); and $dm$::children, which returns the ordered (possibly empty) sequence of children nodes of document and element nodes (and the empty sequence for all other kinds of nodes). The content of a node can be returned as a string value or a typed value (sequence of zero or more atomic values), using the accessors $dm$::string-value and $dm$::typed-value, respectively.

An XML data tree contains a root node, and all nodes that are reachable directly or indirectly from the root by using the $dm$::children, $dm$::attributes, and $dm$::namespaces accessors. The text nodes, which encapsulate XML character content, and attribute nodes are at the leaves of the tree. Each node belongs to exactly one rooted tree. An XML tree whose root node is a document node is referred to as a document, otherwise it is referred to as a fragment.

An XML data tree defines a global document order of nodes, obtained by a pre-order, left-to-right traversal of the nodes in the XML data tree.

Path Expressions

Path expressions are a basic component of XQuery queries. A path expression is composed by one or more steps divided by a “/”. Abbreviations in XQuery are possible: for instance, a sequence of steps may be abbreviated with “//”. 
Each step consists of a forward step or a reverse step or a primary expression and may be optionally followed by one or more predicates. Examples of forward steps are: child::, descendant::, attribute:: etc. An example of reverse step is parent::. A predicate consists of an expression, called a predicate expression, enclosed in square brackets. A predicate serves to filter a sequence, retaining some items and discarding others. The result of the predicate expression is coerced to a boolean value. Those items for which the predicate truth value is true are retained, and those for which the predicate truth value is false are discarded.

Besides predicates, node tests perform a selection based on the type of the node or on the name of the node. For instance, text() selects only text type nodes. A name test only extracts nodes with the same name. Instead of specifying a name, it is possible to specify the wildcard “*” that selects all names. For instance, child::* selects all children of the context item of element type.

The most used abbreviations for path expressions are listed below:

- “.” denotes the current node, i.e. self::node();
- “..” denotes the parent of the current node, i.e. parent::node();
- “//” denotes all descendants of the current node plus the node itself, i.e. descendant-or-self::node();

**FLWOR Expressions**

The name FLWOR is an acronym for the keywords for, let, where, order by, and return.

The basic function of FLWOR expressions is to link one or more variables to expressions results. A variable is indicated by “$” followed by a variable name.

The for and let clauses in a FLWOR expression generate a sequence of tuples of bound variables, called the tuple stream. The where clause serves to filter the tuple stream, retaining some tuples and discarding others. The order by clause imposes an ordering on the tuple stream. The return clause constructs the result of the FLWOR expression. The return clause is evaluated once for every tuple in the tuple stream, after filtering by the where clause, using the variable bindings in the respective tuples. The result of the FLWOR expression is an ordered sequence containing the concatenated results of these evaluations.
3.3. QUERYING THE D²Q MODEL WITH XQUERY

3.3. QUERYING THE D²Q MODEL WITH XQUERY

3.3.2 Quality Functions

XQuery allows users to define new functions. We chose to exploit this possibility in order to access quality data in the DaQuinCIS system. Specifically, we defined a set of XQuery functions, called quality selectors. The defined quality selectors are:

- \( dq: \text{accuracy}(\text{node}^*) \Rightarrow (\text{node}^*) \), taking as input a sequence of nodes and returning a sequence of elements of \( \tau_{\text{accuracy}} \) type corresponding to the accuracy values of each element of the sequence.

- \( dq: \text{completeness}(\text{node}^*) \Rightarrow (\text{node}^*) \), taking as input a sequence of nodes and returning a sequence of elements of \( \tau_{\text{completeness}} \) type corresponding to the completeness values of each element of the sequence.

- \( dq: \text{consistency}(\text{node}^*) \Rightarrow (\text{node}^*) \), taking as input a sequence of nodes and returning a sequence of elements of \( \tau_{\text{consistency}} \) type corresponding to the consistency values of each element of the sequence.

- \( dq: \text{currency}(\text{node}^*) \Rightarrow (\text{node}^*) \), taking as input a sequence of nodes and returning a sequence of elements of \( \tau_{\text{currency}} \) type corresponding to the currency values of each element of the sequence.

- \( dq: \text{quality}(\text{node}^*) \Rightarrow (\text{node}^*) \), taking as input a sequence of nodes and returning a sequence of quality elements containing each four subelements of \( \tau_{\text{accuracy}} \), \( \tau_{\text{completeness}} \), \( \tau_{\text{consistency}} \) and \( \tau_{\text{currency}} \) types corresponding to the quality dimension values of each element of the sequence.

The defined selectors allow to access all available quality data. If for some data nodes there’s no associated quality, then the result is an empty list of nodes.

In Figure 3.11, the implementation of the quality selector \( \text{accuracy}() \) is shown as an example. The function \( \text{searchroot} \) is a function we defined to reach the root of a document containing the input node.

In the following we illustrate some examples of the usage of the accuracy selector. The other selectors work in a very similar way.
CHAPTER 3. THE DAQUINCIS FRAMEWORK: THE D²Q MODEL AND THE DATA QUALITY BROKER

Example 1. Let us consider, in our running example, to ask for the quality of the address of the owner Massimo Mecella. Let us also suppose that quality values are defined for the accuracy and currency dimensions associated to the address, i.e. completeness and consistency are not defined and the symbol $\perp$ populates the corresponding quality properties.

The query is:

```xml
for $i$ in input()//owner[Name eq "Massimo Mecella"]/Address
return quality($i)
```

The result of the query is:

```xml
<root>
  <Address_Quality qOID="qOID132">
    <Address_Accuracy>high</Address_Accuracy>
    <Address_Currency>medium</Address_Currency>
  </Address_Quality>
</root>
```

Notice that: (i) the selector is defined over a set of nodes, therefore if the owner Massimo Mecella had different addresses the quality of each address is returned in the result; (ii) the quality object identifiers have the role of identifying the data object to which quality is associated.

Example 2. In this example, we show that nodes for which quality values are accessed do not need to be homogeneous. Let us consider in our running example to ask for the quality of the address of the owner Massimo Mecella and for the quality of his FiscalCode. Let us suppose that only an accuracy value is associated to the FiscalCode. Let us suppose that only an accuracy value is associated to the Fiscal Code.

The query is:

```xml
for $i$ in input()//owner[Name eq "Massimo Mecella"]
return quality($i/Address,$i/FiscalCode)
```

The result of the query is:

```xml
<root>
  <Address_Quality qOID="qOID132">
    <Address_Accuracy>high</Address_Accuracy>
    <Address_Currency>medium</Address_Currency>
  </Address_Quality>
  <FiscalCode_Quality qOID="qOID131">
    <FiscalCode_Accuracy>high</FiscalCode_Accuracy>
  </FiscalCode_Quality>
</root>
```
3.4 The Data Quality Broker

The Data Quality Broker (DQB) is the core component of the architecture. 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 \(D^2Q\)-compliant \textit{global schema}, specifying requirements 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 requirements.

Quality requirements can be specified on the four data quality dimensions previously introduced, by using quality selectors. For instance, a user can ask for “enterprises with accuracy greater medium”, and the Data Quality Broker selects all the enterprises stored by cooperating organizations, and returns back only those having accuracy greater than the specified threshold.

The Data Quality Broker performs two specific tasks:

- query processing and
- quality improvement.

The semantics of query processing is described in Section 3.4.1, together with an explanatory example. 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 3.4.2. In the following of this Section, the interactions among the internal modules of the broker are shown, while the details on the internal behavior of each module are discussed later in this chapter.

3.4.1 Query Processing

The Data Quality Broker performs query processing according to a \textit{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 [94, 53]. 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. The unfolding of an XQuery query issued on the global schema can be performed on the basis of well-defined mappings with local sources.
The mapping is defined in order to be able to perform a quality improvement function during the query processing step. This means that each concept from the global schema is defined in terms of extensionally overlapping concepts at source, such that when retrieving data a selection can be performed on the basis of the associated quality values. Details concerning how the mapping is actually defined are described later in this chapter.

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 requirements are:

1. let \( Q \) be a query posed on the the global schema \( G \);

2. The query \( Q \) 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 \( Q \) is decomposed in \( Q_1, \ldots, Q_n \) queries to be posed over local sources;

3. The execution of the queries \( Q_1, \ldots, Q_n \) returns a set of results \( R_1, \ldots, R_n \). On such a set an extensional correspondence property is checked, namely: given \( R_i \) and \( R_j \), the items common to both results are identified, by considering the items referring to the same objects. This step is performed by using a record matching algorithm.

4. The result to be returned is built as follows: (i) if no quality requirement 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 3.4.2); (ii) if quality requirements are specified, the result is constructed by checking the satisfiability of the requirements on the whole result.

Example of Query Processing

In the scenario proposed in Section 3.1.1, let us assume that the global schema derived from the INPS, INAIL and CoC local schemas is the one proposed in the Figure 3.12. Some examples of mapping info between global and local schemas are shown in Table ??.

Let us consider the following XQuery query to be posed on the global schema.
3.4. THE DATA QUALITY BROKER

Figure 3.12: Graphical representation of the global schema

<table>
<thead>
<tr>
<th>Global schema</th>
<th>INPS schema</th>
<th>INAIL schema</th>
<th>CoC schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business</td>
<td>Business</td>
<td>Enterprise</td>
<td>Business</td>
</tr>
<tr>
<td>Business/ID</td>
<td>Business/ID</td>
<td>Enterprise/ID</td>
<td>Business/ID</td>
</tr>
<tr>
<td>Holder</td>
<td>Owner</td>
<td>Holder</td>
<td>Owner</td>
</tr>
<tr>
<td>Holder/Name</td>
<td>Owner/Name</td>
<td>Holder/Name</td>
<td>Owner/Name</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>LegalAddress</td>
<td>X</td>
<td>X</td>
<td>LegalAddress</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 3.1: Mapping between global and local schemas
CHAPTER 3. THE DAQUINCIS FRAMEWORK: THE $D^2Q$ MODEL
AND THE DATA QUALITY BROKER

Table 3.2: Query results from the three organizations related to Holder 1

<table>
<thead>
<tr>
<th>Organization</th>
<th>Holder item 1</th>
<th>Accuracy item 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPS</td>
<td>“Maimo Mecella”</td>
<td>Acc_Name = low</td>
</tr>
<tr>
<td></td>
<td>“Via dei Gracchi 71, 00192 Roma, Italy”</td>
<td>Acc_Address = high</td>
</tr>
<tr>
<td>INAIL</td>
<td>“M. Mecalla”</td>
<td>Acc_Name = low</td>
</tr>
<tr>
<td></td>
<td>“Via dei Gracchi 71, Roma, Italy”</td>
<td>Acc_Address = medium</td>
</tr>
<tr>
<td>CoC</td>
<td>“Massimo Mecella”</td>
<td>Acc_Name = high</td>
</tr>
<tr>
<td></td>
<td>“Via dei Gracchi 71, 00192 Roma, Italy”</td>
<td>Acc_Address = high</td>
</tr>
</tbody>
</table>

for $b$ in document("GlobalSchema.xml")//Enterprise
    where $b$/ID ="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 the local schema. The schema mapping information shown in Table 3.1 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 are provided later in this chapter. 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. Tables 3.2 and 3.3 illustrate such results. Each organization returns two holders for the specified business, with the associated accuracy values; in Table 3.2 results concerning one of the two holders are shown, while in Table 3.3 results concerning the other holder are shown. As an example, in Table 3.2, accuracy of properties Name and Address of the first Holder item provided by INAIL are respectively low and medium.

Once results are gathered from INPS, INAIL and CoC, a record matching process is performed. On the basis of the Sorted Neighborhood Method (SNM) [47], we have developed an algorithm for record matching specifically targeted to support the Data Quality Broker query processing; the algorithm is described in Section 4.1. Specifically, we apply the algorithm to data objects returned by an organization, in order to tentatively map them 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.

### 3.4.2 Internal Architecture and Deployment

In this section we describe the high level design of the Data Quality Broker, by considering its interaction modes and its constituting modules; then, an example of query processing execution is shown, by highlighting the roles of the different modules.

#### Interaction Modes and Application Interface

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

- \( \text{invoke(query, mode)} \), for submitting a query \( Q \) over the global schema including quality constraints. The behavior of the broker can be set with the \( \text{mode} \) parameter; such a parameter can assume either the \( \text{BASIC} \) or the \( \text{OPTIMIZED} \) value.

In the basic mode, the broker accesses all sources that provide data satisfying the query and builds the result, whereas in the optimized 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 optimized 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 basic mode, all sources are always contacted; it takes more time to retrieve the final result, but it is always the most accurate one.

Figure 3.13 shows the sequence of operations performed in each of the two modes: in basic mode (Figure 3.13(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 optimized mode (Figure 3.13(b)) only the organization \( \text{Org}_j \) that has the best overall statistical quality evaluation is involved in the query and no comparison is required.

\( \text{propose(data)} \), 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 basic 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 \( R \) selected among the received ones. The \( \text{propose()} \) operation is invoked by a peer instance of the broker in order to provide the quality feedback.

We have currently implemented the Data Quality Broker only in the basic mode; we plan to investigate issues concerning the optimized mode in future work. In the following of this thesis, we will always consider the Data Quality Broker according to the basic mode, no further specifying this.

**Modules of the Data Quality Broker**

The Data Quality Broker is implemented as a peer-to-peer distributed service: each organization hosts a copy of the Data Quality Broker that interacts with other copies. It is internally composed of five interacting modules (Figure 3.14). The modules Query Engine and Transport Engine are general and can be installed without modifications in each organization. These two modules are the ones we have implemented in the DaQuinCIS system and that will be fully described in the next sections. The module Wrapper has to be customized for the specific data storage system. The module Propose Manager is implemented by each cooperating organization according to the policy chosen for taking into account quality feedbacks. The module Comparator, used by the Query Engine in order to compare different quality values, can also be customized to implement different criteria for quality comparison.

**Query Engine** \((QE)\): receives a query from the client and processes it. The query is executed issuing sub-queries to local data sources.
Wrapper \((W r)\): translates the query from the language used by the broker to the one of the specific data source. In this work the wrapper is a read-only access module, data and associated quality stored inside organizations without modifying them.

Transport Engine \((TE)\): communication facility that transfers queries and their results between the \(QE\) and data source wrappers.

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

Comparator \((CMP)\): it is used by the \(QE\) to compare the quality values returned by the different sources to choose the best one.

In the following, we give an overview of the behavior of \(QE\) and \(TE\), leaving to next sections a more detailed description. These are the modules that required most design effort, due to their higher complexity. Indeed, the \(W r\) 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) [49] or Analytical Hierarchy Process (AHP) [82].

The Query Engine is responsible for query execution. The copy of the Query Engine local with the user, receives the query and splits the query in sub-queries (local to the sources). Then, the local QE also interacts with the TE in order to send local queries to other copies of the QE and receive the answers.

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:

- Query execution: the requesting TE sends a query to the local TE of the target data source and asynchronously collects the answers.

- Quality feedback: 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.

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

In this section, we give details of basic 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) 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 3.15 depicts an invocation by a client of the organization Org1, namely:

\[
\text{invoke(for } \$b \text{ in document("GlobalSchema.xml")//Enterprise}
\]
\[
\text{where Accuracy(}$b$/Code eq "APTA") gt "0.5"
\]
\[
\text{return } \langle \text{Holder} \rangle \{ \$b$/Name \} \{ \$b$/Address \}\rangle/BASIC \)
\]

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, QE1 indicates the Query Engine of organization Orgi.

1. QE1 receives the query Q posed on the global schema, as a parameter of the invoke() function; it retrieves the mapping with local sources and performs the unfolding that returns queries for Org2 and Org3 that are passed to TE1;

2. TE1 sends the request to Org2 and Org3;

3. The recipient TE’s pass the request to the Wr’s modules. Each Wr accesses the local data store to retrieve data and quality data;
4. $W_{r_3}$ retrieves a query result, e.g., $R_a$ with accuracy 0.3, and $W_{r_2}$ retrieves another query result $R_b$ with accuracy 0.9;

5. Each $TE$ sends the result to $TE_1$;

6. $TE_1$ sends the collected results ($R_a$, $R_b$) to $QE_1$. $QE_1$ selects through $CMP_1$ the result with the greatest accuracy, i.e., $Org_2$’s result ($R_b$ with accuracy 0.9);

7. $QE_1$ sends the selected result to the client;

8. $QE_1$ starts the feedback process sending the selected result ($R_b$ with accuracy 0.9) to $TE_1$;

9. $TE_1$ sends it to $Org_3$;

10. $TE_3$ receives the feedback data and makes a call $\text{propose}(R_b$ with accuracy 0.9).

## 3.5 Query Engine: The Mediation Function of the Data Quality Broker

The Query Engine module of the Data Quality Broker implements the mediation function of a data integration architecture [103].

In the following sections, we first describe how the Query Engine performs the mapping between the global schema and local schemas (Section 3.5.1). Then, details on how query processing is carried out are provided (Section 3.5.2).

### 3.5.1 Mapping Definition

**Overview and Basic Assumptions**

We have chosen to specify a global-as-view (GAV) mapping between the global schema and local schemas, which means that concepts in the integrated database are described as views over the local databases. The main drawback of a GAV approach with respect to a local-as-view (LAV) approach is that each time a new data source enters the cooperative system, the global schema concepts definition may need to be modified. The DaQuinCIS system has been thought for tightly-coupled information systems, as we will deeply discuss in Section 4.2 of Chapter 4. A principal characteristic of these systems is that they are quite stable in time, therefore the impact of the low modifiability characteristic of GAV systems is reduced in the case of the DaQuinCIS framework.
3.5. QUERY ENGINE: THE MEDIATION FUNCTION OF THE DATA QUALITY BROKER

Figure 3.15: Details of query invocation in the basic mode

(a) Step 1 – 4

(b) Step 5 – 11
In a typical relational environment, a GAV mapping is specified through a set of assertions of the form \(< r, V >\), where \( r \) is a relation from the global schema and \( V \) is a view over the local schema. Each assertion is an association between a relation from the global schema and a query on the local schema.

The DaQuinCIS setting differs considerably from the relational one in that the global and local databases are represented as virtual (non-materialized) XML documents, called respectively global and local views. These views are conform to the \( D^2Q \) model; the global schema and local schemas are expressed as XML Schemas that are also conform to the \( D^2Q \) model.

Differently from what happens in a traditional relational setting, the concept of querying semi-structured integrated data is a quite new field. One of the first example of systems allowing the integration of XML is AGORA [56] though here an extended use of the relational model is still performed. Other examples in which an ontology is used as the global schema are [63, 35].

In the following, we list some simplifying assumptions we have made on the global and local views in the DaQuinCIS framework:

- We do not deal with structural heterogeneities between schemas. This means that if a concept is represented as a class in a schema, this is also represented as a class in another schema. This is a limitation of our model, since in real settings it often happens that structural heterogeneities are present. We plan to remove this limitation in future work.

- We do not have constraints on the schema.

Moreover, \( D^2Q \)-compliant XML Schemas have the following characteristics:

- Elements in a view may have attributes which are used for internal control purposes. These attributes are treated in a way that is completely transparent for the final user. None of the concept relevant for the end user is modelled with an attribute.

- \( D^2Q \)-compliant views don’t contain any comment or processing instruction node, nor any XML entity.

In a semi-structured setting, data corresponding to the same logical concept may be found at different levels of the hierarchy. This implies that it isn’t possible to specify a mapping only by giving correspondences between the concepts specified in the two schemas. It is also necessary to give hints on which is the data location inside documents structure.

Therefore, what we need to build a mapping is a one-to-one correspondence among not only concepts (i.e., data classes) but also possible positions that
their instances may have inside a document. We are helped in this task by the fact that XML views are compliant to the $D^2Q$ model. This imposes a few constraints on the documents’ structure, which can be exploited to simplify the building of a mapping.

An important observation that derives from assumptions above and from the fact that $D^2Q$ schemas are acyclical is that an element that represents a certain type of information, such as a data class, may appear in a fixed number of positions within the document structure. These positions may be enumerated based on the $D^2Q$ schema to which the document corresponds to. As an example, an element representing a data class citizen may be found:

- As a child of the root node, in which case it maps to an object of the class which doesn’t appear as property of another class.
- As a child of an element which has the name of a class having a property whose type is of class citizen.

One of the basic building blocks of all typical languages for semistructured data (and XML) are path expressions (see 3.3.1). Basically, a path expression selects a portion of data based on a description of the path needed to reach it, while browsing the tree hierarchy of the document starting from the root. Although it is possible to query a document without using any path expression, a query style based on this construct is very intuitive and easy; each query can be seen as an expression based on the results of one or more path expressions.

Our idea for translating a query on the global schema is to identify path expressions contained in the query and translate each path expression.

In the following of this section, we provide a formal definition for the DaQuinCIS mapping.

The DaQuinCIS Mapping

The DaQuinCIS mapping is defined by means of a set of tables that establish correspondences among path expressions on local schemas and path expression on the global schema. In order to formally define such a mapping, in the following of this section, we first define the concept of completely specified path expression and the concept of schema element.

XQuery defines two categories of axes to access nodes, namely:

- forward axes, including child::, descendant::, descendant-or-self::, self::, attribute::.
- reverse axes, including parent::.

We can re-arrange this set of axes on the basis of the access to multiple or single level nodes as follows:
multiple-level axes: descendant::, descendant-or-self::.

single-level axes: child::, parent::, self::, attribute::.

Another way to access multiple levels of the document hierarchy is by using the * wildcard to select elements with different names. Steps containing tests based on node kind such as node() or text() also allow the selection of multiple nodes of the same kind, not depending on their names.

Path expressions accessing nodes at multiple levels are a problem, when we want to define a mapping between single concepts: we’d rather like a single path expression to access a single concept from a schema. Here the solution is to transform a path expression containing multiple level axes into an (ordered) list of path expressions containing only single level axes. It is easy to demonstrate that their results can then be concatenated in order to obtain the final result. Note that this transformation is schema-driven: it’s necessary to know in advance which elements can be found in the document hierarchy, in order to expand the steps. To such a scope, we define a completely specified path expression as follows.

Definition 3.5.1 [Completely Specified Path Expression] A completely specified path expression is a path expression that only contains single-level axes. □

Notice that a completely specified path expression (without predicates) does not usually denote a single element, but a set of elements sharing the same path from the root. As an example, the path expression citizen/address/street selects all elements named street which are children of elements named address, which are in turn children of elements named citizen.

Notice that, in general, nodes at the same level correspond to different element types, i.e. to different data classes. As an example, the path expression citizen/address::child() selects all child nodes of addresses, that may include street elements but also city elements. Therefore, we introduce the more restrictive notion of Schema Element.

Definition 3.5.2 [Schema Element] A schema element is any same type element, i.e. corresponding to a \( D^2Q \) data class or property, accessed by a completely specified path expression. □

We are now ready to define a \( D^2Q \) mapping by considering correspondences among schema elements.

Definition 3.5.3 \([D^2Q \text{ Schema Mapping}]\) Let \( e_1 \) and \( e_2 \) be schema elements from two \( D^2Q \)-compliant documents \( S_1 \) and \( S_2 \):
3.5. QUERY ENGINE: THE MEDIATION FUNCTION OF THE DATA QUALITY BROKER

A correspondence $\mathcal{C} : < PE_{e_1}^{S_1}, PE_{e_2}^{S_2} >$ is a couple which associates a completely specified path expression $PE_{e_1}^{S_1}$ identifying $e_1$ within $S_1$, with a completely specified path expression $PE_{e_2}^{S_2}$ identifying $e_2$ within $S_2$.

A Translation Table $T_{S_1S_2}$ between $S_1$ and $S_2$ is a set of correspondences each associating a schema-element from $S_1$ with one from $S_2$, with the additional constraint that a schema-element from one schema may be linked to only one schema element from the other schema. If for each schema-element from $S_i$ a correspondence holds in $T_{S_iS_j}$, then we say that the table $T_{S_iS_j}$ is complete for $S_i$.

Let now $S_g$ be the global schema and $S_{i_1}, S_{i_2}, ..., S_{i_n}$ be the local schemas. A $D^2Q$ mapping $\mathcal{M}$ is a set of translation tables $\mathcal{M}=\{T_{S_gS_{i_1}} \ldots T_{S_gS_{i_n}} \ldots T_{S_gS_{i_n}}\}$ such that $\forall i \in [1 \ldots n]$, $T_{S_gS_{i_n}}$ is complete for $S_{i_n}$.

3.5.2 Query Processing Steps

Query processing is performed according to the sequence of steps described in Figure 3.16.

The Unfolding phase starts by receiving a global query and analyzing it in order to extract those path expressions that access data from the integrated
view (Path Expression Extraction). After a Path Expression Pre-processing step, in the Translation step, based on the mapping, each path expression, expressed in terms of the global view structure, is translated into a set of path expressions over the structure of the local views. The Framing step converts path expressions in a format which is suitable for keeping trace of transformation steps. Finally, queries over local sources are sent to the Transport Engine module, local to the Query Engine, which is in charge of sending queries to local sources to be executed.

The Refolding phase starts with the Re-translation step, in which the received results are re-translated according to the global schema specification. In the next step, i.e. Materialization, results coming from different organizations answering the same global path expression are concatenated into a single, temporary file. Each occurrence of a path expression previously extracted from the global query is replaced with a special path expression that accesses one of the temporary files built during the previous step. In the Global Query Execution step, the global query is changed into a query using only local files, and can then be executed. Then, in the Record Matching, records are matched and after a comparison on quality values externally made by the Comparator Module, the query results ordered by quality are sent for a Quality Filtering. Finally the results best fitting with the user query requirements are sent back to the user. Moreover quality feedbacks are sent to the Transport Engine that is in charge of propagating them in the system.

In the following sections, we give more details on the most complex steps which are the ones of the unfolding phase, namely: Path Expression Extraction, Path Expression Pre-processing, Translation and Framing. In the next chapter, a complete section describes how Record Matching is performed.

### 3.5.3 Path Expression Extraction

The first phase of the unfolding process is the path expression extraction. In the following, we first illustrate how such an extraction works in the basic case. Then, we show an example of how nested path expressions are treated both in the case of nested path expressions including reverse steps and not including reverse steps.

**Basic Case**

Given a query on the global schema, such a query is parsed in order to extract path expressions from it. Only these parts of the query will actually be translated and sent to wrappers for evaluation.

In XQuery, each step of a path expression is evaluated with respect to a context determined by the results of the previous step. A context for the first
step is usually provided by the result of another kind of expression, a variable or a function. The syntax of XQuery offers three functions to access external collections of nodes, namely: `document()`, `collection()` and `input()`. The language specifications allows to define an implementation-specific semantics for the `input()` function. In DaQuinCIS, the `input()` function is used as a mean to access data in the global view. A path expression having the `input()` function as a first step accesses to data integration results. We will refer to these kind of path expressions as to input path expressions.

Path expressions are extracted according to some basic optimizations criteria. As an example, if a path expression is a prefix of another one, then the results of the latter may be obtained by applying to the results of the former all the latter steps not contained in the former. Let’s explain this point with an example.

**Example 3.5.1** Let us suppose that a path expression 1, namely `input()/citizen`, brings to the following result:

```xml
<citizen>
  <name>
    John Smith
  </name>
  <address>
    <street>
      Oxford road
    </street>
    <number>
      26
    </number>
  </address>
</citizen>
<citizen>
  <name>
    Sarah Cardigan
  </name>
  <address>
    <street>
      Cambridge Avenue
    </street>
    <number>
      62
    </number>
  </address>
</citizen>
```

Consider now the following path expression 2: `input()/citizen/address/street`. The results are obviously a subset of that of the first one and could be obtained from it without a second access to the sources. In fact, instead of writing path
expression 2, we could have written the equivalent XQuery expression: let $i=input()/citizen return $i/address/street. □

Therefore, if two path expressions appear in a query and the first one, let’s call it path expression 1, is a prefix of the second one, let us call it path expression 2, then collecting from the sources the results for both of them is a waste of resources. Instead, we can just collect results for path expression 1 and rewrite path expression 2 in a manner such that it selects its own results from those of the first path expression.

Nested Input Path Expressions: Basic Case

The process of path expression extraction may not be straightforward. One case is that of nested input path expressions: a path expression may contain any expression as one of its steps or inside a step’s predicate. Most of the times, such nested expressions can be treated by simply translating any path expression they should contain. In some cases, anyway, a nested expression may contain direct or indirect references to data in the global view. It wouldn’t be correct to just translate such expressions and send them to local sources. Such cases must be handled in a slightly different way. Our current approach is to split any path expression containing a problematic step just before it and to treat the two parts separately.

Example 3.5.2 Let us consider the query:

```
for $i in input()//citizen/owns/building/address
return input()//citizen[livesAt/address eq $i]
```

which (assuming that the address field uniquely identifies a building) returns all citizens who live in a building owned by some citizen. We can extract two input path expressions from this query:

PE1:input()//citizen/owns/building/address
PE2:input()//citizen[livesAt/address eq $i]

Each of them must be separately handled. Nevertheless, while PE1 can be just translated and sent to wrappers, PE2 refers to a portion of PE1’s results through variable $i. It wouldn’t make sense to translate it and send it to organizations as it is, because the translated query would contain an unbound variable. We have instead chosen to split it and just translate the path expression input()//citizen. The predicate selection can be applied to the results of this PE during the global query execution. □
Nested Input Path Expressions with Reverse Steps

The technique followed to execute nested input path expressions may be more complex if reverse steps are included. More specifically, if the remaining part of the path expression contains reverse steps (or other particular functions), it may happen that it tries to access data which haven’t been downloaded. So, each time a path expression is going to be split, it’s necessary to verify if the last part contains any reverse step, in order to appropriately choose where to split.

Example 3.5.3 Consider now this PE:

\[
\text{for } \$i \text{ in input()//citizen/lives_at/address where } \$i = "Oxford Street" \text{ return } \$i/../..
\]

This query returns all citizens living in Oxford street. There are several better ways to write this query, anyway this one is perfectly legal. If we just translated the input path expression herein contained, we would obtain a result composed by citizens’ addresses. Application of the two reverse steps in the return clause of this query wouldn’t lead to a correct answer, because the PE’s result does not contain the nodes which those steps try to select (that is, the citizen nodes). We consider when special conditions of this type occur and we treat them in a special way. For instance, in our example, we send to sources only the translation of the input()//citizen part of the input path expression. □

3.5.4 PE Pre-Processing

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

First of all, notice that a path expression may contain steps written according to an abbreviated syntax. This is not a problem, since it’s always possible to expand such steps to their non-abbreviated correspondents with simple substitutions.

Furthermore, as already mentioned, a simplifying step of the unfolding process is the conversion of path expressions into a completely specified path expressions, that is, all multi-level axes are eliminated.

As we already said, we can transform a path expression containing these kind of constructs into a set of path expressions containing only constructs accessing to a single schema-element; their results may be then concatenated in order to obtain a result exactly equivalent to that of the first path expression. This requires that the document schema is known. As an example, each
Example 3.5.4 For simplicity, we will represent a document schema with a simplified syntax:

```xml
<root>
  <enterprise>
    <VAT/>
    <employee>
      <citizen>
        <name/>
        <surname/>
        <address>
          <building>
            <cityCouncil/>
            <street/>
            <civicNumber/>
          </building>
        </address>
        <owns>
          <building>
            <cityCouncil/>
            <street/>
            <civicNumber/>
          </building>
        </owns>
      </citizen>
      <owns>
        <building>
          <cityCouncil/>
          <street/>
          <civicNumber/>
        </building>
      </owns>
    </employee>
  </enterprise>
  <citizen>
    <name/>
    <surname/>
    <address>
      <building>
        <cityCouncil/>
        <street/>
        <civicNumber/>
      </building>
    </address>
    <owns>
      <building>
        <cityCouncil/>
        <street/>
        <civicNumber/>
      </building>
    </owns>
  </citizen>
</root>
```
Let’s now consider the following path expression.

\[ \text{input()}/\text{descendant-or-self::node()}/\text{child::building}/\text{descendant-or-self::text()} \]

For the sake of conciseness, we will adopt an abbreviated syntax to rewrite the steps. First of all, let’s rewrite the first step. It corresponds to a set of path expressions which point to all nodes which are reachable from the root:

\[
\]
CHAPTER 3. THE DAQUINCIS FRAMEWORK: THE $D^2Q$ MODEL AND THE DATA QUALITY BROKER

enterprise/owns/building/street/text(),
enterprise/owns/building/civicNumber,
enterprise/owns/building/civicNumber/text() )

Then, starting from this sequence, we can treat the step child::building. Note that each path expression that tries to access the element building from a node which does not have a child with that name can be discarded. Only the following path expressions are kept:

(enterprise/employee/citizen/address/building,
enterprise/owns/building)

The third step can now be expanded. We obtain the final sequence:

(enterprise/employee/citizen/address/building/cityCouncil/text(),
enterprise/employee/citizen/address/building/street/text(),
enterprise/employee/citizen/address/building/civicNumber/text(),
enterprise/possiede/building/cityCouncil/text(),
enterprise/possiede/building/street/text(),
enterprise/possiede/building/civicNumber/text())

Note that every member of this sequence is free of unexpanded steps. □

3.5.5 Translation

After such transformations, we can finally translate each obtained path expression into local organizations’ alphabets. Each step is translated on the basis of its name and its position, determined by its preceding steps.

For the sake of simplicity, we suppose to have a canonical $D^2Q$ mapping, i.e. a mapping in which schema elements are specified by a path expression of the type: $a_1/a_2/a_3/.../a_n$ where $a_i$ is a step of the form child::Qname with Qname denoting an identifier compliant to XQuery’s syntax. In this way, if we have no self:: or parent:: step in a path expression, there is a simple path expression that we can easily treat by lookup in one of the translation tables contained in the mapping. If these steps are instead present, we simply need a syntactic rewriting during translation.

We remind that the semantics of the DaQuínCIS system requires that every path expression must be translated with regard to every possible source holding a copy of data relevant to answer the same path expression. Note that, if we cannot find a translation for a certain path expression over an alphabet of a certain organization, it just means that the source does not model that specific concept.

Path expressions may also contain nested subexpressions as steps or within predicates. These expressions may be treated by simply translating any path expression therein found.
3.5.6 Framing

Data received from the organizations must be re-translated to match the global view’s structure. The re-translation process is still based on the mapping defined for the translation process. Notice, anyway, that to translate an element we need to know its simple path from the root.

Typically, the results are sets of nodes with their descendants and don’t contain any information regarding the path leading to those nodes. Nevertheless, observe that the query which leads to the results brings all the information needed to resolve the simple path of the nodes at the top level of the result’s hierarchy. These references can be actually exploited only if results from each single fragment originated from the original query during the phase of constructing completely specified expressions are separated and bound to the fragment itself. To have this property enforced, a solution is to frame those pieces with XML elements, as described in the following. Let us suppose to have a global path expression $PE_g$ and that the translated queries coming from its translation are $PE_1, PE_2, ..., PE_n$. Then the overall query is written as:

\[
<1> PE_1 < /1 > \\
<2> PE_2 < /2 > \\
... \\
<n> PE_n < /n > .
\]

The results are:

\[
<1> resultofPE_1 < /1 > \\
<2> resultofPE_2 < /2 > \\
... \\
<n> resultofPE_n < /n > .
\]

and then it is immediate to associate them to the related fragment. This introduces, of course, the need to treat in a special way the returning of the quality document associated to those results.

When all results relative to a same global path expression have been re-translated, they can be concatenated into a unique result, discarding the framing tags.

3.5.7 Query Engine Implementation

In this section, we provide an overview of the implementation of the Query Engine. In Figure 3.17, the main implementation modules of the Query Engine are shown; the phases of query processing shown in Figure 3.16 that are executed by each module are also represented.

The Query Engine has been implemented as a Java application. The following sections describe the main implementation choices. Details on the
implementation of the Record Matcher are provided in the next chapter. The Quality Filtering module is trivial, therefore no details on it are provided.

**Query Parser**

To manage parsing of XQuery queries, a parser has been generated with the help of the JavaCC (Java Compiler Compiler) tools, which are a suite of applications for the generation of lexical and syntactic analyzers in Java. The generation of a parser starts with the specification of a grammar file, which contains a description of the tokens and of the grammar rules used by the language for which the parser is built. A JavaCC grammar file for XQuery is available from W3C for testing purposes. The file has been re-edited to adapt the generated parser to our needs. Specifically, some grammar rules have been completely re-built in order to support the identification of input path expressions. The proposed grammar file leads to a parser which builds single-mode syntactic trees, i.e. trees for which a single class exists which models the concept of node. With single-mode syntactic trees, different kind of nodes are distinguished by a node-id. Anyway, JavaCC offers a useful framework that makes easier to visit a tree using *visitor* classes obtained by subclassing a single interface. This framework only works with multi-mode syntactic trees, with a distinct class for each node appearing in the grammar. To take advantage of the *visitor* framework, the grammar file was modified so to obtain a multi-mode parser. Specific visitor classes are used to perform many tasks involved in the unfolding phase.

During unfolding, a first syntactic tree is built by parsing the query. The
tree is then annotated by computing the bindings of each variable appearing in the query. Then, what we call a *path expression equivalent* for the query is computed. A path expression equivalent is a path expression that accesses the same set of data in the global view as the query. Note that the term *equivalent* does not mean that the result of this path expression would be identical to that of the original query. What we do is simply substituting some expressions, which are not path expressions, with path expressions that access the same data (or a superset of them), though maybe in a different order. This step is performed in order to support the identification of nesting in input path expression and in order to detect the presence of variables depending on input path expressions. As noted before, these cases must be handled by splitting input path expressions. The result of this step may contain expression sequences, that is lists of expressions separated by commas. Such a result is changed into a vector of expressions which does not contain any expression sequences. Each of the resulting expressions is then re-parsed and its syntactic tree is analyzed to identify any input path expression. This step includes ad-hoc treatment of path expressions requiring to be split. The resulting set of input path expressions is reduced by eliminating any duplicates and path expressions whose results are *included* by other path expressions. As already explained, the results of a path expression are included into another one’s results if the latter is a prefix of the former.

Each input path expression is associated to a unique file name, which is assigned to the temporary document built from its results. The global query is then rewritten by substituting any occurrence of each one of the input path expressions with a new path expression which uses the `document()` primitive to extract the whole content of the temporary file associated to it.

**Translator/Re-Translator**

A single package contains classes implementing both translation and retranslation functionalities. These functionalities are based on a mapping which is, as already described, a collection of translation tables, one for each organization participating to the DaQuinCIS system. Translation tables can be added or removed according to modifications of the set of organizations taking part to the DaQuinCIS system. They are actually submitted as XML files. Each translation table contains the correspondence between the global schema and a local schema, as a dictionary which allows to translate between two simple path expressions, in both directions.

In the translation procedure, extracted input path expression identified during the unfolding step are passed to a method which takes care of all translation steps. The package offers functionalities to translate path expressions into completely specified path expressions. After this splitting, each result-
CHAPTER 3. THE DAQUINCIS FRAMEWORK: THE $D^2Q$ MODEL AND THE DATA QUALITY BROKER

ing path expression can be translated. Translation is attempted with each translation table existing in the mapping. Translations (where they exist) are then framed (as explained in the previous section) and returned to the caller, arranged in a data structure which holds information about the original input path expression and the target alphabet of each translation.

The retranslation phase is still performed by translation package methods. Each result is composed by two documents: a data document, which contains the answer to the query, and a quality document, which contains data quality information on that answer. These two documents contain elements that must be retranslated. The retranslation step is quite straightforward, as it only requires that the two document are visited and that each element name is substituted with its translation over the global schema alphabet. While descending the document’s hierarchy, a memory must be kept of which path is followed, in order to identify each element. Note that a context for the topmost nodes in the document is provided by the query which originated the result. Indeed, if the query is a framed collection of queries, correspondence is still guaranteed as the framing is replicated in the result. Once retranslation has been carried out, the framing tags can be safely eliminated.

Re-translated results relative to a same input path expression (coming from different organizations) are then concatenated and written into a temporary file (which can be discarded after executing the global query). The name of the file is that which was previously (during unfolding) associated to the input path expression. In an analogous manner, the re-translated quality document fragments are concatenated and written into temporary files whose names are dependant on those used for their corresponding data documents. Note that, to avoid conflicts between OIDs and quality OIDs coming from different organizations, each OID contains a code which uniquely identifies the originating source.

IPSI-XQ Query Engine

Once extracted from a query, path expressions are substituted with new expressions pointing to the results of the former path expression. This pointing is obtained through the document() function included in XQuery.

Example 3.5.5 Consider the query:
\begin{verbatim}
for $i$ in input()//citizen/owns/building/address
return accuracy($i)
\end{verbatim}

It only contains one input path expression, which is identified during the unfolding phase, translated and sent to wrappers. Each wrapper returns a result to the path expression, with an associated quality document. Results (and quality documents) are retranslated and concatenated to form a single
answer, then written on a temporary file. Suppose that the name of this file is `resultdoc.xml`. In order to execute the global query, it has to be rewritten in the form:

```xml
for $i$ in document("resultdoc.xml")/*//node()
return accuracy($i).
```

Executing the global query is now immediate. It can be just passed to a query engine. The query engine used in this implementation is IPSI-XQ [1]. Let us note that we needed to make IPSI-XQ quality-aware by adding the defined quality functions (see Section 3.3.2). These functions are written in XQuery, and simply added to the query prolog of each query submitted to the engine.

### 3.5.8 Transport Engine

The Transport Engine (TE) component provides the connectivity and communication infrastructure of the DaQuinCIS system. In Figure 3.18 the internal components of the TE are shown, as well as their interactions.

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

---

**Figure 3.18:** Internal modules of the Transport Engine of organization i
The *Transport Engine Interface* is the module that interfaces the Query Engine and the Transport Engine. Specifically, it uses a data structure to store queries as well as query results, once they have been gathered by each organization. The data structure is organized as an array: each element is representative of a single query execution plan and is composed by a list of queries that are specific of such a plan. Such queries are passed by the *Query Engine* (step 1). Then, the *Transport Engine Interface* activates the *Execute-Query* module with plans as input parameters (step 2).

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

Notice that the same interaction among modules shown in Figure 3.18 occurs when quality feedbacks have to be propagated. The *Query Engine* interacts with the *Comparator* in order to select the best quality copies among the ones provided as query answers and then sends back the result to the *Transport Engine Interface* that activates the *Execute-Query* module with the best quality copies and the organizations to be notified about them as input parameters. The best quality copies are then sent by the *Web Service Invoker*. On the receiver organization side, the *Execute-Query* module notifies the *Propose Manager* modules of involved organizations about the better quality data available in the system, thus implementing the quality feedback functionality that the Data Quality Broker provides at query processing time.

Notice that the *Execute-Query* module, on the sender organization side, also interacts with the *Availability Tester* modules: this makes quality notification not to be performed in a one-step process. Instead, a transaction starts that commits only when the set of sources that has to be notified is exhausted.
Chapter 4

The DaQuinCIS Framework: The Record Matcher, The Rating Service and Other Modules

This chapter fully describes two modules of the DaQuinCIS system, namely:

- the Record Matcher [12], describing how matching records in presence of quality data;
- the Rating Service [29], describing how trusting sources with respect to provided data.

The Record Matcher is described in Section 4.1. Instead, the Rating Service is described in Section 4.2.

In Section ??, we provide an overview of two further modules of the DaQuinCIS framework, namely: the Quality Notification Service [58, 57] and the Quality Factory [23]. We simply describe the main functionalities of these two modules, without specifying their detailed design which is out of the research scope of this thesis.

4.1 The Record Matcher

The quality improvement policy underlining the architecture shown in Figure 3.1 mainly involves two components, namely: the Data Quality Broker and the Quality Notification Service. The Data Quality Broker realizes improvement
each time a query with quality requirements is posed, thus performing an (on-line improvement). The Quality Notification Service maintains quality levels by diffusing information on quality updates.

The Data Quality Broker works according to two sequential phases: (i) linkage of different copies of the same data available in the CIS; (ii) improvement on the basis of best quality copies. As seen in the previous sections, best quality copies are selected in the query processing phase by the Query Engine that interacts with the Comparator; the Transport Engine is then in charge to invoke the Propose Manager for making organizations to implement quality improvement actions. The linkage phase is instead carried out by the Record Matcher module.

In order to design and implement the Record Matcher, we have studied the more general problem of matching records in a system where quality data are attached to each data value.

We propose to adopt a method for record matching based on the quality data exported by cooperating organizations. The proposed method is based on the Sorted Neighborhood Method (SNM) [48]. As describe in Section 2.3.1, the SNM which consists of three distinct steps:

- Choice of the matching key. The choice of the key is a fundamental step of the matching process, as the results of the matching are directly influenced by it.
- Sorting of records according to the chosen key.
- Moving of a fixed size window through the list of records and comparisons only of the records included in the window. Each couple of records in the window is compared in order to decide if the two records match or not.

Our proposal is different from SNM with reference to the following aspects:

- Automation of the matching key choice phase. This phase is typically realized by a human expert, known as key designer. Instead we suggest using quality data exported by organization in order to select such a key. The details of the proposed key selection algorithm are described in Section 4.1.1.
- The decision about the matching of two records, i.e. the duplicate detection, is taken in a domain independent way by considering a function that normalizes a classic edit distance function upon string lengths. Such a function and the procedure used to declare that two records are duplicate are described in Section 4.1.2.
4.1. THE RECORD MATCHER

4.1.1 Automatic choice of the matching key

We propose to exploit quality data exported by each cooperating organization in order to automatically choose the matching key. The idea is to choose a high quality key in terms of accuracy, completeness and consistency. We have not considered currency as this dimension has a quite different way to be measured, with respect to the other dimensions; nevertheless, we plan to add currency in future work.

Let us consider as an example the choice of a key with a low completeness value; after a sorting on the basis of such a key, the potential matching records can be not close to each other, due to null values. Similar considerations can be made also in the case of low accuracy or low consistency of the chosen key; a low accurate or low consistent key does not allow to have potential matching records close to each other. Therefore, quality of data is considered as an important feature in choosing the matching key. In Section 4.1.1, the procedure followed to evaluate the quality of attributes candidate as potential keys is explained.

Besides quality of data, the other element influencing the choice of the key is the identification power. In the following of this Section, the exact definition of the identification power is provided together with the final formula according to which choosing the matching key.

Evaluating quality for choosing the matching key

According to $D^2Q$ model, for each exported data value some quality values are associated to it. In this Section, we will first show how to calculate an overall value of completeness, accuracy and consistency for each attribute, starting from the values of such dimensions for the attribute values in all records. Then we calculate an overall quality value for each attribute by combining computed dimension values on the basis of an empirical function.

We focus on three dimensions, i.e. accuracy, consistency and completeness. Moreover we assume that the result of the evaluation of each of these dimensions for a data value may be of three types, namely: YES (Y), NO (N) or INAPPLICABLE (I). We associate to each data value a triple of quality values corresponding to the values of each of the chosen dimensions. In the following we assume, without loss of generality, that the triples are ordered, i.e. the value in the first position corresponds to the completeness value, the value in the second position corresponds to the accuracy value, and the value in the third position corresponds to the consistency value. The possible values for the triples are the leaves of the tree shown in Figure 4.1; the tree is constructed starting from completeness, and considering the subsequent values of accuracy and consistency. As an example, if completeness=N, i.e. the value is absent,
the accuracy and consistency values are both INAPPLICABLE (right side leaf in Figure 4.1).

We associate a numerical value to the values Y, N and I of the three dimensions, according to Table 4.1. These numerical values allow to weight positively or negatively the quality values of attributes. Notice that for consistency we divide for the number \( M \) of attributes considered in the consistency check. In the following of the paper, we consider for simplicity \( M = 2 \). According to such numerical values, the quality triples shown in Figure 4.1, can be ordered as represented in Table 4.2.

The overall value of completeness, accuracy and consistency, for each attribute \( j \) is the mean related to the total number of values of such attribute. As an example, if \( N \) is the total number of values of the \( j \)-th attribute, the completeness value is:

\[
\text{Completeness}_{j} = \frac{\sum_{i=0}^{N} \text{completeness}_{i,j}}{N}
\]

In order to evaluate an attribute \( j \) as a potential key, an overall quality
4.1. THE RECORD MATCHER

<table>
<thead>
<tr>
<th>Value</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYY</td>
<td>+3</td>
</tr>
<tr>
<td>YYI, YIY</td>
<td>+2</td>
</tr>
<tr>
<td>YYN</td>
<td>+1.5</td>
</tr>
<tr>
<td>YNY,YII</td>
<td>+1</td>
</tr>
<tr>
<td>YIN</td>
<td>+0.5</td>
</tr>
<tr>
<td>YNI</td>
<td>+0</td>
</tr>
<tr>
<td>YNN</td>
<td>-0.5</td>
</tr>
<tr>
<td>NII</td>
<td>-1</td>
</tr>
</tbody>
</table>

Table 4.2: Numerical values for dimension measures

The value need to be calculated for it.

We introduce a function **Data Quality of the attribute** \( j \) \( (dq_j) \) that combines quality values of each dimension:

\[
dq_j = \alpha \times \text{completeness}_j + \beta \times \text{accuracy}_j + \gamma \times \text{consistency}_j
\]

The values \( \alpha, \beta \) and \( \gamma \) have been determined according to the results of the experiments on data matching. The values \( \alpha = 0.45, \beta = 0.45, \gamma = 0.10 \) are the ones that have produced the best performance metrics of the record matching algorithm (see Section 4.1.3).

Notice that it is also possible to consider a set of attributes as candidate for the matching. In such cases, we consider the Data Quality of the set as given by the arithmetic average of the Data Quality values of single attributes.

**Identification Power**

The quality of data does not adequately characterize attributes candidate as matching key. Let us considering as an example a record Citizen with fields **Surname**, **Name**, **Address** and **Sex**. Though the field **Sex** may have the greater quality value, it would not be appropriate as matching key because it can have only two values, i.e. **Male** and **Female**, and thus all records are simply divided into two sets, without having similar records close to each other.

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

**Definition 4.1.1 [\( eq_j \) relation]** Given two records \( r1 \) and \( r2 \), and given an attribute \( j \) of the two records, we define the equivalence relation \( eq_j \) such that
**r1 eqj r2** iff **r1.j=r2.j**, i.e. the value of the attribute j of the record r1 is equal to the value of the attribute j of the record r2. □

**Definition 4.1.2 [ipj identification power]** The Identification Power of the attribute j ipj is defined as the number of distinct equivalence classes originated by the relation eqj applied on the total of records ÷ Total number of records □

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

Another important observation is that the identification power is not sufficient by itself to choose the best key for matching. If the values of an attribute are affected by many errors, the identification power may be high only because of such errors that originate many distinct classes.

**Key choosing function**

In the previous sections, we have shown that quality of data and identification power, if considered independently, are not sufficient to choose a good quality key.

In this section we introduce a function that combines these two parameters in order to choose the key for matching.

If dqj is the overall quality value and ipj is the identification power, defined in Section 4.1.1, we introduce the function kj, such that:

\[ k_j = dq_j \times ip_j \]

Let us consider all the attributes j of records, the steps to calculate the matching key are thus the followings:

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

In case of selection of a set of attributes to construct the key, the computation of the Data Quality and the Identification Power varies according to the extensions described in Section 4.1.1.
4.1. THE RECORD MATCHER

4.1.2 Duplicate detection

The method we propose for duplicate detection is based on a specific edit distance function; string or edit distance functions consider the amount of difference between strings of symbols. We have chosen the Levenshtein distance \[54\], which is a well known early edit distance where the difference between two text strings is simply the number of insertions, deletions, or substitutions of letters to transform one string into another.

The function we use for deciding if two strings \(S_1\) and \(S_2\) are the same is also dependent from the lengths of the two strings as follows:

\[
f(S_1, S_2) = \frac{\max(\text{length}(S_1), \text{length}(S_2)) - LD(S_1, S_2)}{\max(\text{length}(S_1), \text{length}(S_2))}
\]

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

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

- the function \(f\) is applied to the values of a same attribute in the two records. If the result is greater than a fixed threshold \(T_1\), the two values are considered equal; we call \(T_1\) field similarity threshold.

- If the number of equal pairs of values in the two records is greater than a threshold \(T_2\), then the two records are considered as duplicates; we call \(T_2\) record similarity threshold.

The thresholds \(T_1\) and \(T_2\) have to be fixed experimentally. In section 4.1.3, some examples are given on how to consider the values of such thresholds on our experimental data set.

4.1.3 Experiments and Results

We have tested the proposed method on databases generated with two different automatic database generators. The first generator, that we call G1, is the same used in \[48\]. G1 has some relevant limitations, namely: impossibility of null values in the main attributes and poor variability in the set of possible values of generated databases. We have thus implemented a second generator, that we call G2, in order to remove such limitations.

G2 allows to choose among the following database fields: Social Security Number, Surname, Name, Birth Date, Birth City and Province, Sex, Address, City, Province, ZIP and Profession. The field’s values are selected from
the following lists (in brackets lengths are shown): male names (1799), female names (4674), surnames (88799), cities (8072), provinces (99) and ZIP’s (11782), professions (6318) and address (18719).

It is worth noting that G1 produces databases that are quite far from real ones, as an example names, surnames and addresses are generated from the same list. Instead G2 provides for a greater field value variability. G2 also allows to select null values for each field thus generating databases that are more difficult to treat but also closer to real ones.

The performance metrics used in this work are **precision** and **false positive**.

**Precision** is: \[
\frac{\text{Number of detected correct duplicates}}{\text{number of total duplicates in the database}}
\]

**False positive** is: \[
\frac{\text{Number of incorrectly detected duplicates}}{\text{number of total detected duplicates}}
\]

These metrics are the same used in [48].

We have performed two sets of experiments. The former is related to the proposed algorithm tested on databases generated by G2. The latter compares the performances of our algorithm with the results published in [48], and thus is performed on databases generated by G1.

The former set of experiments is described in the Figures from 4.2 to 4.5, and has been performed in order to choose values for the thresholds T1 and T2. According to what introduced in Section 4.1.2, T1 the field similarity threshold and T2 the record similarity threshold. In the first pair of figures, namely Figures 4.2 and 4.3, the precision and false positive metrics are measured for
three values of T1, while T2 changes. In the second pairs of figures, namely Figure 4.4 and 4.5, the situation has been inverted, i.e. T2 values are fixed, while T1 changes. From such experiments, it is possible to derive that to maintain a precision greater than 70%, T2 has to be lower than 7 (see Figures 4.2 and 4.4). Moreover, a too much low level for T2 improves very little the precision, while making the false positive to decrease (see Figures 4.4 and 4.5). On the other hand, in order to have low percentages for false positive, values for T1 should be greater than 0.4 (see Figure 4.3). For the following experiments the chosen values are for T1 is 0.85, and for T2 is 0.5.

The latter set of experiments runs the proposed method on the database created by the generator G1. The results are compared with the results obtained by forcing our algorithm on one of the key chosen in [48], with the scanning window dimension changing. The results are shown in Figures 4.6 and 4.7. Two notable results are obtained by an automated choice of the matching key:

1. the matching key chosen in an automatic way is almost independent from the size of the scanning window, i.e. also for small values of such a size the performances are high;

2. the matching key chosen in an automatic way has better performances as far as precision values, with a considerable improvement for low values of the scanning windows (about 70% vs. 40% of the human chosen key); as for false positive values, the human chosen key produces better
Figure 4.4: Precision vs. Field Similarity Threshold T1 on a database generated by G2

performances, i.e. lower false positive values, but results are quite similar (both under 0.05%).

Most notably, comparing the performances of our algorithm with the results published in [48], our algorithm reaches precision percentages ranging between 70% and 80% even for low values of the scanning window, i.e. belonging to the [0,10] interval. Conversely, according to the results published in [48], in the same conditions, precision ranges between 60% and 70% on a single pass, i.e. by considering a single key sorting. Also false positive values are always lower than the ones published in [48], on a single pass (notice that in the previous sets of experiments, our algorithm is run with two different matching keys and results are compared; instead, the results above described compare our algorithm with the traditional SNM).

We have not yet implemented a multi-pass approach, consisting of making a transitive closure with different keys, but obtaining better results on a single pass approach makes us suppose that better results will be also obtained with a multi-pass approach.
Figure 4.5: False Positive vs. Field Similarity Threshold T1 on a database generated by G2
Figure 4.6: Precision vs. Scanning Window on a database generated by G1
Figure 4.7: False Positive vs. Scanning Window on a database generated by G1
4.2 Ensuring Trust in DaQuinCIS: the Rating Service

When organizations composing a CIS exchange data each other, a relevant problem is how to trust each organization with respect to the quality of provided data.

In this section, we describe a trust model and its implementation as the Rating Service module of the DaQuinCIS architecture; the reader can refer to [29], for more details.

We propose a model for trusting cooperating organizations, in which a trust value is assigned to each organization with respect to a specific data category. The trust level is assigned on the basis of a probabilistic model that considers the satisfaction of the receiving organization in a data exchange. We consider examples taken from the Italian e-Government scenario, where public administrations interact as peers in order to fulfill service requests from citizens and enterprises. In such a scenario, our method allows for assigning a trust level to administrations or private companies taking part to the cooperation. Therefore, for instance, in the Italian e-Government scenario the Department of Finance can have a high trust level with respect to Fiscal Codes of Citizens and a low trust level with respect to their Residence Addresses. The proposed model also allows to fix a threshold to discriminate between trusted and untrusted organizations.

For loosely coupled P2P environments, some examples of reputation systems have been introduced, in order to manage trust issues. The Rating Service module can be seen as a reputation system which is targeted to the DaQuinCIS system and more generally to tightly coupled P2P systems (loosely and tightly coupled systems are explained later in this section).

Different motivations can be provided for proposing a reputation system also for tightly coupled P2P systems, namely:

- In tightly coupled P2P systems, it is very difficult to individuate a subject that is responsible for a certain data category. In fact, data are typically replicated among the different participating organizations, and one does not know how to state that an organization has the primary responsibility for some specific data, i.e. it is the Data Steward [80] of that specific category. By proposing an evaluation of organizations’s trust with respect to the quality of provided data, we give a mean for defining a data steward as the organization having the highest trust level with respect to a specific data category.

- For tightly coupled P2P systems of limited dimensions and bounded lifecycle, the use of trust techniques can be very useful as it dramatically
ensuring trust in DaQuincis: the rating service

reduces the typical start-up phase in which a peer needs to become confident versus other peers.

Many other significant benefits can derive from our proposal. First, as our model is thought for tightly coupled environments that are typically implemented by cooperative processes, such processes can be designed on the basis of the determined reputation on provided data. Second, the cooperation itself is strengthened by enabling to know organizations’ reputation: data are more often asked to peer organizations if a trust value for them is available. As an example, in our e-Government scenario, administrations very often prefer asking citizens for data, rather than other administrations that have already stored the same data, because they do not know if trusting them or not.

The remaining of this section is organized as follows. We propose a possible classification of P2P systems in Section 4.2.1. Then, we introduce a novel model for trust in Section 4.2.2 which has been implemented by an architectural service detailed in Section 4.2.4. In Section 4.2.5, we illustrate some experimental results.

4.2.1 DaQuinCIS as a Peer-to-Peer System

The interest for peer to peer systems has been considerably growing in the last years. Although P2P systems are considered a revolution in network based environments, they are actually only an evolution of the original internet model, enabling packet exchanges among nodes with interchangeable roles.

With the P2P acronym, we indicate each distributed system in which nodes can be both clients and servers. In other words, all nodes provide access to some of the resources they own, enabling a basic form of interoperability. Many simple P2P applications are today widely spread, like file sharing and instant messaging. Moreover, more complex applications, such as distributed processing, begin to be available.

A large number of systems, having different objectives and functionalities, could be included in the basic definition given above. An interesting classification of P2P systems can be found in [3]. The following three models of P2P systems are introduced:

Decentralized Model. The applications belonging to this class can be considered as pure P2P systems. The architecture is completely decentralized, without common elements shared by peers. Each peer may have a knowledge of the system by exchanging information with other peers.

Centralized Model. P2P systems belonging to this class have at least one common element, such as a peer search index. This introduces a bottleneck, though allowing peers to look for information more accurately and quickly.
Hierarchical Model. This class could be considered as an intermediate class between the other two. There is a set of nodes, called super-peers, that assume particular functions such as peer address index or local control.

Such a classification is based upon the level of distribution of control and access functions, such as centralized or decentralized indexing. We extend this classification by adding another dimension, that is orthogonal to the other. This dimension takes into account how much *tight* can be a peer interaction. For example, with systems such as Kazaa [52], the interaction is not tight because users can only search for data and establish temporary connections. Indeed, with distributed workflow systems, each node can have more sophisticated and long interactions with the other nodes, thus originating a tight interaction. As an example, PARIDE [59], a framework that enables dynamic composition and orchestration of web services in peer to peer systems, allows complex, even transactional, process interactions between peers.

We classify P2P systems into the following two classes, on the basis of their interaction level:

- **Loosely** coupled systems, consisting of systems in which data exchanges are not pre-defined. Data exchanges simply imply a request/reply interaction between peers.
- **Tightly** coupled systems, based on pre-defined processes that cross organizational boundaries. Such processes can even be transactional thus originating a very tight interaction among the different organizations that participate to the system. All data exchanges are placed within a defined process.

Combining these categories with the ones described in [3], we can obtain a more complete description of P2P systems, based on behavioral characteristics. For example, Self-Serv [38] belongs to the class of P2P decentralized models and has a tightly coupled interaction model; conversely, Gnutella [44] belongs to the decentralized class, and has a loosely coupled interaction model. In the matrix of Figure 4.8 we report the classification of some relevant proposals with respect to both illustrated dimensions. In the following of this section,

<table>
<thead>
<tr>
<th></th>
<th>Loosely Coupled</th>
<th>Tightly Coupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decentralized Model</td>
<td>Gnutella [44], Kazaa [52]</td>
<td>Self-Serv [38]</td>
</tr>
<tr>
<td>Hierarchical Model</td>
<td>FastTrack [37]</td>
<td>PARIDE [59]</td>
</tr>
<tr>
<td>Centralized Model</td>
<td>Napster [68]</td>
<td>DaQuincis [87]</td>
</tr>
</tbody>
</table>

Figure 4.8: Examples of P2P systems classification
we concentrate on tightly coupled P2P systems (P2P-TCS’s) by providing a trust model for the DaQuinCIS architecture.

4.2. Trust Model: Basics

In defining a trust model for the DaQuinCIS architecture, we exploit some peculiarities of tightly coupled systems. Notice that the techniques for ensuring trust in P2P systems proposed by the current literature mainly concern loosely coupled systems. Examples of such techniques can be found in [66, 27, 3].

In P2P-TCS’s, a first important difference with respect to loosely coupled ones, is the complete knowledge of the identity of peers involved in data exchanges. The possibility to maintain an anonymous identity is common in peer to peer applications. However, malicious agents can exploit this weakness to spread undesired contents or dangerous applications, like virus. The use of IP addresses to avoid these behaviors was proposed [27], but this solution is poorly efficient; for example, spoofing techniques or use of dynamic IP address and proxies servers can easily make the method unreliable. The organizations that take part to P2P-TCS’s are instead known to each other and this reduces the probability of fraudulent behaviors. Nevertheless, we can’t exclude that an organization has a dishonest behavior. For example, let us suppose that organizations \(\text{Org}_i\) and \(\text{Org}_j\) have the same kind of data \(\mathcal{D}\). If \(\mathcal{D}\) are somewhat crucial for \(\text{Org}_i\)’s processes, \(\text{Org}_i\) could try to discredit \(\text{Org}_j\) to get a kind of data stewardship on \(\mathcal{D}\) data.

Another important difference with loosely coupled systems, is the dynamism of the system, in terms of the number of participating organizations. In loosely coupled P2P systems, the number of peers changes frequently, due to the typical anonymous and transient nature of interactions. Conversely, P2P-TCS’s organizations typically have a stable process-based interaction.

These differences have two main implications on the trust model, namely:

- a more specific trust model can be considered, based on the definition of trust with respect to a category of data;
- misbehavior may be involuntary, e.g. due to temporary problems. Therefore, once an organization is classified as untrusted, it cannot remain in this state indefinitely, but specific reinstatement methods need to be provided.

When deciding the atomic unit to trust, a first hypothesis could be to trust the organization as a whole, with respect to the totality of exchanged data or more generally to the transactions performed with other organizations. The method proposed in [3] is an example of this case.
We follow the approach of associating trust to an organization as a whole but we propose two major modifications, namely: (i) we consider a specific type of transaction, i.e. data exchanges; (ii) we evaluate trust of an organization with respect to a specific type of provided data.

More specifically, a trust value is associated to a couple \( < \text{Org}_i, \mathcal{D} > \) where \( \mathcal{D} \) is a data unit (see below). In this case we have a finest granularity level of trust on the sources. Organizations can choose different partners for data exchanges relying onto a wider range of possibilities in sources’ selection. As an example, in the Italian e-Government scenario, it may happen that the Department of Finance is recognized as trusted with respect to Fiscal Codes, but untrusted with respect to Residence Addresses (of Italian citizens). Therefore, other administrations can ask mainly to the Department of Finance to obtain Fiscal Codes.

Before describing our trust model, we provide some basic definitions:

**Organizations.** They are considered with respect to the role of providing data each other and consuming data from each other. Notice that in such a way they are peers, i.e. they can have both roles of data consumer and data provider. Furthermore, we suppose that organizations are independent and have a competitive behavior.

**Data Unit.** A data unit can be a generic view on data provided by organizations. As an example, a class Citizen, or a single attribute, such as Name of Citizen.

**Source Unit.** A source unit is a couple \( < \text{Org}_i, \mathcal{D} > \), where \( \text{Org}_i \) is the organization providing the data unit \( \mathcal{D} \).

**Complaint.** Given \( \text{Org}_i \) sending a data unit \( \mathcal{D} \) to \( \text{Org}_j \), \( \text{Org}_j \) can raise a complaint \( C_{i,\mathcal{D}} \) stating that \( \mathcal{D} \) data are low (not satisfactory) quality data.

Complaints are used to calculate trust of source units.

In [3], a method to evaluate trust of sources in P2P systems is described. In this work, the authors introduce the following formula:

\[
T(p) = | \{ c(p, q) \mid q \in P \} | \times | \{ c(q, p) \mid q \in P \} |
\]

where \( P \) is the set of the agents and \( c(p, q) \) represents a complaint raised by agent \( p \) if it believes agent \( q \) is cheating. The decision on the agent’s trustworthiness is done on the basis of \( T(\cdot) \) values. Low values indicate that the peer is trustworthy.

The method we propose to evaluate trust of source units is inspired by the one above, but with some major differences.
4.2. ENSURING TRUST IN DAQUINCIS: 
THE RATING SERVICE

The first difference is related to the atomic unit of trust, that in our case is the couple $<\text{Org}_i, \mathcal{D}>$. 

The second basic difference develops over an intuition described by the following example. Let us suppose that a set of organizations requires a typology of data $\mathcal{D}^k$ 100,000 times to $\text{Org}_j$, and that 1000 complaints are fired by the same set. For the sake of simplicity, we suppose that only $\text{Org}_j$ has $\mathcal{D}^k$ data. This should be interpreted as an indication that $\mathcal{D}^k$ are good quality data, because only a small fraction of it originated complaints. Notice that the function $T(\cdot)$ calculated on the couple $<\text{Org}_j, \mathcal{D}^k>$ is equal to 1000. Moreover, let us suppose that after 1000 accesses to data $\mathcal{D}^h$ provided by $\text{Org}_i$, 100 complaints are fired. The $T(\cdot)$ function of the couple $<\text{Org}_i, \mathcal{D}^h>$ is equal to 100. On the basis of $T(\cdot)$ values, we should consider $<\text{Org}_i, \mathcal{D}^h>$ as trusted and $<\text{Org}_j, \mathcal{D}^k>$ as untrusted. The decision algorithm proposed in [3] does not take into account the number of requests sent to a source unit, and run correctly only if each peer makes approximately the same number of transactions; in this case, in fact, $T(\cdot)$ values are really comparable.

In the following section, we provide a method that considers the overall number of data exchanges with respect to which complaints are raised.

4.2.3 Definition of a Trust Parameter for Source Units

The trust level of a source unit is calculated on the basis of the number of complaints fired by other organizations. Let us assume that $\mathcal{O}$ is the set of peer organizations and that $\mathcal{C}_{i,j,\mathcal{D}} = C(\text{Org}_i, <\text{Org}_j, \mathcal{D}>)$ is the complaint that $\text{Org}_i$ fires with respect to the source unit $<\text{Org}_j, \mathcal{D}>$.

As previously discussed, the number of complaints is not sufficient by itself to evaluate source units’ trust. Therefore, we need a mechanism to store the number of requests made to each source unit. We propose to associate such information to each complaint, thus defining the following structure for messages exchanged within the P2P system in order to guarantee trust management:

$$\mathcal{C} = <\text{Org}_i, <\text{Org}_j, \mathcal{D}>, n>$$

where $\text{Org}_i$ is the organization which has fired the complaint and $<\text{Org}_j, \mathcal{D}>$ is the source unit the complaint is referred to. The integer $n$ represents the number of requests of $\mathcal{D}$ issued by $\text{Org}_i$ to $\text{Org}_j$, starting from the last complaint that $\text{Org}_i$ fired against $\text{Org}_j$.

Furthermore, we assume that $n_{i,j,\mathcal{D}}$ is the number of complaints that $\text{Org}_i$ sends to $<\text{Org}_j, \mathcal{D}>$. We introduce the following source unit’s trust parameter:

$$R(<\text{Org}_j, \mathcal{D}>) = \frac{\sum_i \mathcal{C}_{i,j,\mathcal{D}}}{\sum_i n_{i,j,\mathcal{D}}} \quad \forall \text{Org}_i \in \mathcal{O}$$
The numerator represents the overall number of the complaints issued by organizations with reference to the data unit $<\text{Org}_i, \mathcal{D}>$. The denominator is the overall number of interactions in which the data unit $\mathcal{D}$ is sent by $\text{Org}_j$ to other organizations.

High values of $R(\cdot)$ mean that the source unit is not trustworthy. We suppose that each interaction could be modelled as a random variable of a probabilistic process, that can have value 1 if a complaint is fired, 0 otherwise. Specifically, we introduce the random variable $X$ such that:

$$X = \begin{cases} 
1 & \text{if a complaint is fired} \\
0 & \text{otherwise}
\end{cases}$$

We suppose, without loss of generality, that variable $X$ has a binomial probability distribution with $P(X = 0) = p$ and $P(X = 1) = 1 - p$. Moreover, we make the following basic assumption: $p \ll (1 - p)$. In fact it is reasonable to suppose that the number of unsuccessful interactions is low with respect to the number of successful ones.

Therefore, $R(\cdot)$ is a random variable, linear combination of a large number of independent random variables that have the same distribution of probability. Thus, due to the “Central Limit Theorem”, we can suppose that $R(\cdot)$ has a normal probability distribution.

A Criterion for Trust. The calculation of $R(\cdot)$ values allows one to establish a sorting of source units on the basis of their trust values. This is a novel contribution with respect to the current literature, that simply allows for discriminating between trusted and untrusted sources, without establishing a rating within these two sets. Instead, such a rating can be very important because an organization can choose to request data from another organization which has not the best trust value (though being trusted) but for example has a higher response time.

Nevertheless, it is important to distinguish between trusted and untrusted organizations. Therefore, we need to introduce a threshold, such that one can easily identify which organizations can be trusted for data with a fixed probability. A threshold also allows for applying a policy for punishing untrusted organizations. When $R(<\text{Org}_j, \mathcal{D}>)$ exceeds the threshold value, the source unit is automatically excluded from the system.

We argued that $R(\cdot)$ is a normal distributed variable. Then, we can calculate the mean $m$ and the standard deviation $\sigma$ and, by exploiting properties of the normal distribution, we give the following trust criterion:

$$\text{IF } R(<\text{Org}_j, \mathcal{D}>) \leq m + 2 \cdot \sigma \text{ THEN } <\text{Org}_j, \mathcal{D}> \text{ TRUSTED}$$
$$\text{ELSE } <\text{Org}_j, \mathcal{D}> \text{ UNTRUSTED}$$
According to the properties of the normal distribution, such criterion ensures that the right number of samples is selected with a probability at least equal to 95%.

The hypothesis of the same distributions of $X$ variables for the different organizations could be removed. In this case, the given criterion continues to be valid on the basis of the Tchebycheff inequality [32], i.e.:

$$P(|X - \mu| > K \cdot \sigma) < \frac{1}{K^2}$$

where $\mu$ is the mean and $\sigma$ the variance of $R(\cdot)$ samples. In the case of a normal distribution, the inequality is exactly reduced to the given criterion for $K = 2$. In the general case, the Tchebycheff inequality simply guarantees a probability lower bound of 75%.

Managing Malicious Behaviors

Cheating for Data Stewardship. In the hypothesis of our system, it may happen that an organization sends a large number of complaints in order to discredit another organization on the same data it owns; in this way, it can conquer the stewardship on such data. In order to prevent from such a malicious behavior, we introduce an adjusting term in the trust parameter definition, as follows:

$$R(< O_{rgj}, D >) = \frac{\sum_i C_{i,j,D} + \sum_i C_{j,i,D}}{\sum_i n_{i,j,D} + \sum_i n_{j,i,D}} \forall O_{rgi} \in O$$

The second term of the numerator penalizes organizations requiring data they already have to other organizations and firing a large number of complaints. The term at the denominator corresponds to the term introduced to take into account the number of interactions.

Bias Elimination. We adopted for the $R(\cdot)$ computation the following rule. If an organization has fired a very high percentage of complaints against a data unit then we do not consider them in $R(\cdot)$ calculation. In fact, this situation is due either to strict requirements on data quality or to malicious behavior. The fraction complaints-interactions, we call $F(c, i)$, is a normal random variable (see Section 4.2.2). If we focus on the $F(c, i)$ values of an organization, we first calculate the parameters (i.e. the mean and the standard deviation) of the normal distribution of the other organization $F(c, i)$ values. Then, if $F(c, i)$ is higher than the mean value plus two times the standard deviation, the organization’s complaints are not included in the $R(\cdot)$ calculation.
4.2.4 Rating Service Design

In this section, we describe the design of the Rating Service module of the DaQuinCIS architecture, which implements the trust model described in the previous section.

The Rating Service has a centralized architecture storing all complaints and is responsible for certifying the trust level of source units. The use of a centralized system does not give specific scalability problems, due to the low variability of the number of source units in P2P tightly coupled systems. Moreover, the presence of a trust service independent from each organization could be a disincentive to have a malicious behavior. The Rating Service provides a complete knowledge of organization’s behaviors with respect to the quality of data they exchange. This is a further advantage with respect to the fully decentralized peer to peer systems, which have a limited knowledge of complaints on a single peer.

**Calculation of $R(\cdot)$**. We assume that each organization in the system tracks its interactions with other organizations. In such a way, the Rating Service can simply gather this information, and calculate the overall number of transactions on some specific data made by an organization, necessary for source units’ trust evaluation.

The Rating Service works as follows. When an organization is not satisfied by a source unit, it sends a complaint to the Rating Service. The Rating Service stores such a complaint and updates the statistics about the source unit.

The Rating Service analyzes the source unit’s complaints and interaction numbers and calculates the values of $R(\cdot)$ for each source unit. The frequency of trust values update is a parameter that the Rating Service administrator should fix. If the quality of the organization data changes frequently, the associated trust value should be often updated.

Organizations can query the Rating Service in order to have the trust level of a source unit. Then, they can use such values to select a source unit, either in combination with other quality metrics, or simply by selecting the ones having the best (i.e., the smallest) values of $R(\cdot)$.

**False Values of $n$.** Notice that malicious organizations could send complaints with false values in the interaction field, in order to make another organization untrustworthy. In the current Rating service implementation, we suppose that the values in the interaction field are correct, leaving to future work the treatment of fraudulent behaviors concerning this field. Here, we simply give some hints on a possible way to solve the problem of false values
of $n$ in the complaints. Organizations can query the Rating Service to obtain source units’ trust levels. However, they cannot see complete information about single organization exchanges. Let us assume that organizations track all received requests. This is easy to achieve; for example, current DBMS’s maintain such a statistic information. When an organization supposes that its trust level does not conform to the value provided by the Rating Service, it can ask for a check procedure, sending the number of interactions it stored for each organization. Then, the Rating Service requests the last interaction number to organizations present in the source unit table. In such a way, the Rating Service interaction number values are directly comparable with data sent by the organization. The comparison of such values allows one to verify if an organization cheats on complaints.

4.2.5 Experimental Validation of the Model

We have verified the effectiveness of the trust model described in the previous section, by implementing the rating service within the DaQuinCIS architecture. We use the Colt distribution library [26] for the statistical functions and results analysis.

The results obtained from the performed set of experiments confirm the reliability of our trust model. As a first set of experiments, we evaluate the ability of the Rating Service to identify those organizations having a malicious behavior or very strict requirements on data quality. We call this type of organizations anomalous organizations. We create one source unit and three sets of respectively 25, 50 and 75 organizations that require data from the source unit. Each organization poses a different number of data requests; the number of requests ranges from 150 to 600. We perform several simulation runs, varying the number of organizations having an anomalous behavior. The experimental results are shown in Figure 4.9. Notice that when the number of anomalous organizations exceeds 15% of the total number of organizations, the performance of our Rating Service degrades rapidly. This result shows the importance of assuming the number of positive interactions much greater than the number of unsuccessful interactions.

In the second set of experiments, we prove the effectiveness of the trust decision criterion. We create 25 source units that export 14 different typologies of data units and 75 organizations requiring data. The number of requests varies between 150 e 600. We introduce an organization that cheats for data stewardship and one source unit that provides data generating an high number of complaints. The value of trust assigned by the Rating Service to each source unit is reported in the chart of Figure 4.10. The dashed line represents the mean of the trust values calculated by the Rating Service. Instead, the continuous line represents the threshold values of trustworthiness (see Section
4.2.3). Source unit 12 is a source unit providing poor quality data, and source unit 20 refers to an organization cheating for data stewardship. The results show that the Rating Service identifies correctly both untrusted source units.
4.3 The Other Modules: Quality Notification Service and Quality Factory

4.3.1 Quality Notification Service: Overview

The Quality Notification Service (QNS) is the component of the DaQuinCIS system used to inform interested users when changes in quality values occur within the CIS. This component is used to control the quality of critical data, e.g., to keep track of quality changes and to be always aware when quality degrades under a certain threshold. The interaction between the Quality Notification Service and its users follows the publish/subscribe paradigm: a user willing to be notified for quality changes subscribes to the QNS by submitting the features of the events to be notified for, through a specific subscription language. When a change in quality occurs, an event is published by the QNS i.e., all the users which have a consistent subscription receive a notification. However, as discussed in [58], currently available pub/sub infrastructures do not allow to meet all the requirements that a QNS implementation should satisfy, in particular scaling to a large number of users and coping with platform heterogeneity.

The Quality Notification Service addresses both these problems through a layered architecture that (i) encapsulates the technological infrastructure specific of each organization, (ii) adopts the standard Web-service technology to implement inter-organization communications, and (iii) embeds solutions and algorithms (namely, merge subscriptions and diffusion trees) to reduce the use of physical and computational resources. The interested reader can find more details on the QNS in [87]. The QNS is no further described here as it implements distributed techniques, that are out of the scope of this thesis.

4.3.2 Quality Factory: Overview

The Quality Factory is the component of the DaQuinCIS architecture that is in charge of managing data quality assessment activities inside each cooperating organization. The Quality Factory is mainly composed by two software modules, namely: a Quality Analyzer module and a Quality Assessment module. The Quality Analyzer, by using quality measurement tools, performs a static analysis of quality dimensions values associated to data exported to other cooperating organizations; such values are compared with benchmark quality parameters contained in a Quality Repository. If data values do not satisfy intra-organizational quality requirements, they have to be sent to the Quality Assessment module, which improves the level of data quality to allow the complete or partial fulfillment of quality requirements. Notice that, as it will be better explained in Chapter 5, internal requirements can also be partially
satisfied but data are exported in any case. Indeed, here the approach is to admit that such data can be useful to other organizations in the CIS.
Data quality improvement is an issue inherently very complex. The proof of this complexity can be found in the various methodologies dealing with it that have been published both by researchers (e.g., [96, 88]) and practitioners (e.g., [34, 80]). When considering data quality improvement in cooperative contexts, things become even more complex: problems specifically related to data quality are coupled with problems coming with the cooperative contexts, such as different data settings, different responsibilities and ownerships on data etc.

We claim that a first fundamental step when wishing to deal with quality issues in cooperative systems is to perform an intra-domain analysis in order to understand data and processes within which data are exchanged. A further objective of such an analysis is to understand quality requirements that organizations would like to specify on data. Finally, a first high level hypothesis on how the quality of data could be improved in the system is also among the “desiderata” of such an analysis.

On the basis of our experience in a real setting, namely the Italian e-Government one, a complete methodology is needed in order to carry out only this preliminary analysis step aimed at understanding data quality requirements and possibilities in terms of data quality improvement. In Section 5.1 of this chapter we describe the IP-UML methodology, the methodology we propose for data and quality analysis and for designing high level quality improvement actions [86].

Nevertheless, we are convinced that dealing with data quality issues in a cooperative setting needs to be supported by a software architecture that
automatically manages data exchanges and implements improvement functionalities. The DaQuinCIS framework has been thought with these ideas in mind and is placed within a complete methodology that extends the Total Data Quality Management (TDQM) [96] to the CIS’s context, namely the TDQM_CIS methodology (in [13] the preliminary ideas). We describe the TDQM_CIS methodology in Section 5.2, showing how the DaQuinCIS system fits within all the related methodological phases.

In Section 5.3, we discuss how IP-UML and TDQM_CIS can be integrated in a complete methodology for quality improvement in CIS, namely the DaQuinCIS methodology.

Finally in Section 5.4, we describe an experience in the Italian e-Government setting, in which the IP-UML methodology was applied. We are also currently testing the TDQM_CIS methodology by using the same data sets described in 5.4; the preliminary results of such validation experiments will be available for the final version of this thesis.

5.1 The IP-UML Methodology

In the research field, some proposals for analysis methodologies aiming at understanding possible data quality problems have been proposed, e.g. [96, 88]. When adopting one of these methodologies, a major problem is the lack of a modeling language that can help throughout all the methodology phases. Such a language should be formal enough to allow a unique interpretation of the language constructs, i.e. a precise semantics should be defined for it. At the same time, it should be easily understood also by people who do not have scientific and technical skills; this is because one of the most important activities when designing for quality improvement is the interaction with customers in order to find out the actual quality requirements. In the software engineering area, a language that has both characteristics is the Unified Modeling Language (UML). In fact, UML has a defined semantics and perhaps the principal cause of its success lies in its understandability even for non-technical people. However, the UML semantics is intentionally general. This is because different application domains and systems need different specialization and the applicability of UML would have been constrained if it could not support such diversity. The specialization of UML to different domains has already been realized by the proposals of different profiles, i.e. extensions of UML for specific purposes. Some examples of these profiles can be found in [71], and are the "UML Profile for Software Development Processes" and the "UML Profile for Business Modeling".

In this Section, we will first describe a proposal for a UML profile for data quality. This profile is inspired by the concepts of the IP-MAP framework,
proposed in [88]. The IP-MAP framework is an extension of the Information Manufacturing System (IMS) proposed by Ballou et al. in [8]. IP-Maps have the major advantage of combining both data analysis and process analysis in order to assess quality of data. Data are considered as a product, called Information Product (IP), and the processes creating data are analyzed in order to point out quality bottlenecks in manufacturing information products. The IP-MAP framework already proposes a modeling formalism, which is based on Data Flow Diagrams. The use of UML instead of such a formalism has the following advantages: UML is a standard language and a lot of CASE tools for it have been implemented; UML is a language supportive of analysis, design and implementation artifacts, so the same language can be used in all the phases of the process, thus bridging the gap between the initial phases of an improvement process and the actual implementation; finally, the expressive power of UML is greater, especially with reference to the process modeling constructs, as detailed in the following sections.

Later in this Section, we place the UML profile for data quality within a methodology that is based on the IP-MAP framework but differs from this framework because: 
(i) it specifies the artifacts to produce during the improvement process in terms of diagrams drawn by using the UML elements defined in the profile for data quality; 
(ii) it uses IP-MAPs not only in order to assess quality and to think of improvement actions but also as a diagrammatic way to design improvement actions.

More specifically, the following sections are structured as follows. In Section 5.1.1, a brief overview of the IP-MAP framework and of UML extension mechanisms is described. In Section 5.1.3, the UML profile for data quality is proposed. Section 5.1.7 and 5.1.8 describe a UML and IP-MAP based methodology for quality analysis.

5.1. IP-MAP Framework Basics and UML Extension

An Information Production Map (IP-MAP) is a graphical model designed to help people to comprehend, evaluate, and describe how an information product such as an invoice, customer order, or prescription is assembled. The IP-MAP is aimed at creating a systematic representation for capturing the details associated with the manufacturing of an information product. IP-MAPs are designed to help analysts to visualize the information production process, identify ownership of the process phases, understand information and organizational boundaries, and estimate time and quality metrics associated with the current production process. There are eight types of construct blocks that can be used to form the IP-MAP. Each construct block is identified by a unique and non-null name and is further described by a set of attributes (metadata). The content of these metadata varies depending on the type of
construct block. In Table 5.1, the possible types of construct blocks are shown, together with the symbol used for their representation.

**UML Extension Mechanisms**

The Unified Modeling Language (UML) is used in many different application domains and is increasingly becoming the de facto standard language for object-oriented analysis and design; now UML is at version 1.5 [71]. UML definition is very general so that it can be used in all kinds of application domains; in addition, it can be extended in order to cope with peculiar aspects of specific systems and domains. Some standard extension mechanisms are provided, namely: constraints, tag definitions and tagged values, and stereotypes. The specification of UML analysis and design elements is based on the
5.1. **THE IP-UML METHODOLOGY**

notion of a model element, defined as an abstraction drawn from the system being modeled; the principal model elements are classes and relationships.

A *constraint* is a semantic restriction that can be attached to a model element; it can be expressed: *(i)* in an informal language, when the interpretation must be done by a human; *(ii)* in a formal language, in order to be automatically interpreted. In UML diagrams, constraints are enclosed in braces.

A *tag definition* specifies new kinds of properties that may be attached to model elements.

A *tagged value* specifies the actual values of tags of individual model elements.

A *stereotype* is a model element that characterizes model elements through a precise semantics.

According to the UML specification, ”a coherent set of such extensions, defined for a specific purpose, constitutes a UML profile”.

In the following sections, we focus on stereotypes and constraints as extension mechanisms, in order to define a UML profile for data quality improvement. The profile, described in Section 5.1.3, is based on the IP-MAP framework and its main objective is to give a formal definition to the main concepts related to the management of quality improvement. Moreover, the profile also allows to organize such concepts into a set of models useful to implement a system for quality improvement.

### 5.1.2 A Running Example

We consider an example first described in [77]. For the purposes of this example, the information product is constituted by mailing labels. A school called Big State University uses these mailing labels to send out publications to its alumni. Incorrect or out-of-date mailing labels are a problem for the university. After the end of each academic year, data (including current address information) about graduating seniors are taken from the Big State University’s active student database and transferred to the Alumni database. Alumni are encouraged to send name/address corrections and changes to Alumni Affairs so that their address information can be kept up-to-date. The secretary at Alumni Affairs records this information into the Alumni database on a weekly basis. Unfortunately, only about 1 in 10 alumni remember to inform Big State University of their name and address changes. When it is time for Big State University to send out an alumni publication, Alumni Affairs runs a program to create a list of mailing labels, which are then pasted onto the outgoing publication by the University Mail Service. The goal of Big State University is to improve the quality of mailing labels, in order to reduce the percentage of undelivered mail due to out-of-date or incorrect mailing labels.
5.1.3 Data Quality Profile

In this section, the Unified Modeling Language is extended in order to allow for modeling and designing data quality improvement actions. The starting concepts are the ones defined in the IP-MAP framework; the result of such an extension is a UML profile called Data Quality profile.

The Data Quality profile consists of three different models, namely: the Data Analysis Model described in Section 5.1.4, the Quality Analysis Model described in Section 5.1.5 and the Quality Design Model described in Section 5.1.6. The three models derive from the idea of improving the quality of data by: (i) identifying which data are of interest, what is their composition and what is their derivation in the context of the production process (Data Analysis Model); (ii) identifying the quality requirements for each data type (Quality Analysis Model); and (iii) modeling data and processes together in order to verify quality requirement satisfaction and to model process-based improvement actions (Quality Design Model).

5.1.4 Data Analysis Model

The Data Analysis Model specifies which data are important for consumers, as their quality is critical for organizations’ success. The IP-Map framework considers: (i) information products, i.e. results of processes manufacturing data items; (ii) raw data, i.e. parts of information products, and (iii) component data, i.e. semi-processed information that contribute to the manufacturing of the information product itself. In the Data Analysis Model, each of these elements is represented as a stereotyped UML class, namely:

- << informationProduct >> . An Information Product class is a class labeled with this stereotype that represents an information product.
- << rawData >> . A Raw Data class is a class labeled with this stereotype that represents a raw data related to a specific information product.
- << componentData >> . A Component Data class is a class labeled with this stereotype that represents a component data related to a specific information product.

We also introduce a further stereotyped class:

- << qualityData >>. A Quality Data class is a class labeled with this stereotype that generalizes Information Product classes, Raw Data classes, and Component Data classes.

The relationships among these elements are shown in the schema depicted in Figure 5.2; this is a meta-schema, i.e. a schema defining the elements to be
5.1. THE IP-UML METHODOLOGY

used in the UML schemas modeling the application systems. In the figure, an Information Product class is an aggregation of Raw Data classes and it has a dependency relationship with Component Data classes, which means that if component data change, therefore also the information product will change. A dependency relationship also connects the Component Data class with the Raw Data class. Moreover, a Quality Data class has a generalization relationship with Information Product classes, Raw Data classes and Component Data classes. Notice that the concepts of information products, raw data and component data are all defined in the IP-MAP framework; instead, the concept of quality data has been introduced in order to provide a higher abstraction level when generally referring to data "interesting" from a data quality perspective, i.e. for which quality requirements may be specified. Moreover, the UML formalization of such concepts also helps to understand the relationships and dependencies among the different elements.

5.1.5 Quality Analysis Model

The Quality Analysis Model consists of modeling elements that allow representing quality requirements of data. A quality requirement can be related to one of the quality dimensions or characteristics that are typically defined for data quality.

Without loss of generality for the model, for which ad-hoc sets of dimensions can be defined, we consider the set of dimensions introduced in Section 3.2, namely: accuracy, completeness, consistency and currency.

Our idea is to model the overall set of dimension-related requirements as a hierarchy of stereotyped classes, all of which are subclasses of a Quality Requirement class. The following stereotypes need to be introduced:
Figure 5.3: Classes and associations in the Quality Analysis Model

- **<< qualityRequirement >>**. A Quality Requirement class is a class labelled with this stereotype and generalizing the set of quality requirements that can be specified on a Quality Data class.

- **<< qualityAssociation >>**. A Quality Association is an association relationship labelled with this stereotype and associating a Quality Requirement class with a Quality Data class. Quality requirements on data need to be verified, such that, in case they are not satisfied, improvement actions can be taken; therefore a constraint is specifically introduced on the Quality Association.

The meta-schema defining the elements of the Quality Analysis model is shown in Figure 5.3. Specifically, the hierarchy of stereotyped classes specializing a Quality Requirement class is depicted as well as the Quality Association relating a Quality Requirement class and a Quality Data class. The specification of a distinct stereotype for each quality requirement has the advantage of clearly fixing the types of requirements that can be associated to data. Moreover, in future work the possibility of considering a measure class for each requirement (in a way similar to the one proposed in [90]) will be investigated. Notice that we have also considered the opportunity of formalizing constraints by using the Object Constraint Language (OCL)\(^1\) [102]. We decided not to adopt OCL because it is inherently too much complex to be used in high level analysis methodology. Instead, our idea is to provide a modeling language fully understandable "at a glance", without requiring too much effort in the study phase. An example of a diagram of the Quality Analysis Model related to our running example is shown in Figure 5.4; the accuracy, currency and completeness quality requirements are specified respectively on the Quality Data classes Mailing Label, Name and Address.

\(^1\)The Object Constraint Language (OCL) is part of the UML and can be used to specify all kinds of constraints, pre- and post-conditions, guards etc. over the objects in the different models.
5.1. THE IP-UML METHODOLOGY

5.1.6 Quality Design Model

The quality design model specifies IP-MAPs. An IP-MAP helps to understand the details associated with the manufacture of an information product. It shows the processes managing data, as well as how information products are produced starting from raw data and component data. Such a combined process and data analysis allows to achieve two objectives: (i) to detect quality potential problems and, after having checked the non-conformance to quality requirements, (ii) to introduce quality improvement actions.

The IP-MAP dynamic perspective, in which processes are described together with exchanged data, can be obtained by combining UML activity diagrams with UML object flow diagrams.

Activity diagrams are a special case of state diagrams in which “all (or at least most) of the states are action or subactivity states and in which all (or at least most) of the transitions are triggered by completion of the actions or subactivity in the source states” [81].

Object flows are diagrams in which “objects that are input or output from an action may be shown as object symbols. A dashed arrow is drawn from an action state to an output object, and a dashed arrow is drawn from an input object to an action state” [81].

The following UML extensions need to be introduced, in order to represent IP-MAP elements:

- Stereotyped activities, to represent processing and data quality blocks.

Figure 5.4: A diagram of the Quality Analysis Model of the running example
Figure 5.5: Stereotypes to model IP-MAP constructs

- Stereotyped actors, to represent customer, source, and data storage blocks.
- Stereotyped dependency relationships, in order to give a precise semantics to the relationships between some elements.

In Table 5.5, the detail of all the stereotypes with the associated descriptions is listed.

Notice that:

- The Decision Block, included in the IP-MAP specification, is simply the standard decision construct of UML activity diagrams.
- The Information System Boundary Block and the Business Boundary Block are represented by the standard concept of swimlanes.\(^2\)

The IP-MAP as described in [88] is very similar to Data Flow Diagrams; it is also based on the same approach, i.e. it has a specific focus on data. Instead, some characteristics specifically related to processes, such as process synchronization, also need to be captured when modeling for quality improvement. UML activity diagrams and object flows allow on one hand to maintain the IP-MAP idea of combining data and processes, but also enrich the process modeling power by exploiting the synchronization constructs included in the

\(^2\)“Actions and subactivities may be organized into swimlanes. Swimlanes are used to organize responsibility for actions and subactivities” [81]
5.1. THE IP-UML METHODOLOGY

Figure 5.6: A diagram of the Quality Design Model of the running example

activity diagrams specification. In Figure 5.6, an example of a diagram of the Quality Design Model related to the running example is shown.

5.1.7 IP-UML: a Methodology for Data Quality Analysis

We propose a methodology for data quality analysis, called IP-UML. A methodology definition includes the specification of [89]:

- what modeling language should be used in describing the analysis and design work;
- a development process, that is a set of rules which defines how a development project should be carried out;
- some techniques concerning how to produce the project artifacts (e.g., analysis model, design model, documents, etc.).

As to now, we have focused on the modeling language and the development process, leaving to future work possible techniques supporting the process itself.
CHAPTER 5. THE DAQUINCIS METHODOLOGY: IP-UML+TDQM_CIS

128

Figure 5.7: Phases of the process with artifacts

IP-UML uses the Data Quality profile, defined in the previous section, as the modeling language. A process to design quality improvement actions is described in the following section.

5.1.8 A Process to Improve Data Quality

We propose a process consisting of three distinct phases, iteratively executed: Data Analysis, Quality Analysis and Quality Improvement Design, as shown in Figure 5.7.

The Data and Quality Analysis phases are inspired by the IP-MAP framework, and are simply detailed by the specific UML artifacts that should be produced in each phase in conformance to the Data Quality profile. The Quality Improvement Design phase consists of two distinct sub-phases, namely: the Quality Verification phase and the Quality Improvement phase. The former is inspired by the IP-MAP framework; while, the latter has been introduced with the specific aim of using IP-MAPs explicitly modeling improvement processes.

Data Analysis Phase

The Data Analysis phase implies the analysis of the data that are exchanged, stored, updated, modified by processes of a single organization (or information system) or of multiple organizations (or information systems). The main focus is on data that are “critical” for organizations success, as a poor quality for them can cause costs in terms, for example, of lost sales or penalties when the quality must be guaranteed according to contractual requirements. The result
of data analysis phase is the *data analysis model*, introduced in Section 5.1.4. Such a model consists of a set of *class diagrams* describing identified information products and their composition in terms of raw data and component data, modeled by the Data Quality profile.

### Quality Analysis Phase

The Quality Analysis phase specifies the required quality, in terms of quality dimensions, for data identified in the Data Analysis phase. For each dimension, the Data Quality profile introduces stereotyped classes associated to a Quality Data class, according to what defined in Section 5.1.5. The result of this phase is a *quality analysis model*, which consists of a set of *class diagrams*, describing Quality Data classes with the specified quality requirements.

### Quality Improvement Design Phase

The Quality Improvement Design phase can be divided into two sub-phases:

- **Quality Verification**
- **Quality Improvement**

  In the Quality Verification phase, the process by which the information product is manufactured is described. Moreover, if quality bottlenecks of such a process are detected, quality checks can be introduced in order to directly face the problem by controlling data in the weakest process phases. The result of this sub-phase is a set of *activity diagrams*, in which object flows are described in order to model data exchanges and manipulations. The activity diagrams are drawn by using the stereotyped activities, actors and relationships introduced in Section 5.1.6.

  Though the Quality Verification phase solves quality problems in many cases, more complex quality improvement actions often need to be engaged, involving not simply the introduction of control points, but a complete process re-engineering driven by quality requirements. In the Quality Improvement phase, improvement actions are defined to such a scope. Specifically, this phase consists of a level design of processes realizing quality improvement.

  As in the Quality Verification phase, activity diagrams and object flows are drawn by using the constructs introduced in Section 5.1.6. Diagrams of both the Quality Verification and the Quality Improvement phases constitute the *quality design model*. 

5.2 TDQM_CIS

Based on the analogy between data and manufacturing products, an extension of Total Quality Management (TQM) to data is proposed in [96], called Total Data Quality Management (TDQM). The process underlining this methodology considers four phases as necessary for managing the Information Product (IP): definition, measurement, analysis and improvement. These phases are iteratively executed, thus constituting a cycle. The definition phase includes the identification of data quality dimensions and of the related requirements. The measurement phase produces quality metrics that provide feedback to data quality management and allows the comparison of the effective quality with pre-defined quality requirements. The analysis phase identifies the roots of quality problems and studies their relationships. The improvement phase consists of information quality improvement techniques.

We have redesigned the four phases of the TDQM methodology in the context of cooperative systems, thus giving rise to a TDQM_CIS cycle. The definition, measurement and improvement phases are preserved, though with a semantics which is specific of the cooperative setting. Instead, the analysis phase is replaced by an exchange phase that takes into account the need for exchanging and comparing data in a cooperative setting. In Figure 5.8, the TDQM_CIS cycle is shown.

In the following of this Section, we provide details on each of the phases of the TDQM_CIS, by focusing on how the DaQuinCIS system supports the actual implementation of the individual phases.
5.2. **TDQM.CIS**

5.2.1 **The Definition Phase**

A fundamental requirement for organizations to cooperate is the knowledge about data that each organization exports and makes available to the other ones. In addition to this, we claim that it is necessary for organizations also to export the quality of their data. The $D^2Q$ model, described in Section 3.2, has been proposed to such a scope. Indeed, it defines data schemas, quality schemas and the associations between them.

We recall that within the IP-UML methodology, the data analysis model and the quality analysis model provide an analysis on data categories of interest and on the associated quality dimensions.

The definition phase of TDQM.CIS takes as input the sets of diagrams included in the data analysis model and in the quality analysis model of the IP-UML methodology, and produces as output:

- XML local Schemas of cooperating organizations, which need to be $D^2Q$-compliant;
- an XML global Schema, which is also $D^2Q$ complaint.

Notice that the class diagrams of the data and quality analysis model are quite straightforward translated into $D^2Q$ schemas, in which the basic concept is a data class.

Notice also that in the translation we do not represent the notion of quality requirement, which is instead used in the measurement and improvement phases to drive related actions.

5.2.2 **The Measurement Phase**

Each organization in the CIS has to perform an internal quality measurement activity in order to evaluate the quality of its own data asset.

The result of such an assessment has to be compared with the quality requirements specified by the quality analysis model of the IP-UML methodology. Notice that such requirements are specified on data owned by a specific organization, though they can be data managed by internal processes and by cooperative processes too.

If quality requirements are not satisfied, quality improvement actions need to be internally engaged by the organizations. Also in the case that internal quality requirements are not satisfied, the quality of data that are exchanged with other organizations has to be exported; indeed, such data can satisfy strictless quality requirements of other organizations.

In the following, we give an extended definition of the specific quality values that need to be exported.
In the $D^2Q$ model definition, we made the assumption that each leaf of the quality graph contains a “single metadatum” consisting of the value of a dimension for a specific data class or property. This allowed us to focus on model properties without losing generality.

Generally speaking, a single value representing a data quality dimension evaluation has no meaning in a cooperative context, if we do not further specify it. Therefore, we will not stay anymore with a single metadatum, but we introduce a “set” of quality metadata characterizing each data item.

Specifically, we propose to describe these quality values using a general metadata format that accommodates a rating, that is the results of a measurement, and that explicitly includes the description of the criteria adopted for the measurement. Note that this approach conforms to standard practices used in various scientific fields when presenting experimental results (e.g., the result of a medical test is presented along with the specific technique used to carry out the test).

The metadata descriptor is a set of triples, of the form:

$$[\text{qualityRating}, \text{RatingRangeDescriptor}, \text{RatingMethodDescriptor}]$$

where each triple carries the quality rating for a specific dimension. Each triple consists of the following elements:

- QualityRating is the quality value for the dimension, expressed as a number (a Real) in the normalized range 0-100;
- RatingRangeDescriptor describes the meaning of the quality values, i.e., what the rating number stands for;
- RatingMethodDescriptor describes the criteria used to obtain the rating, i.e., how the measurements are carried out.

Unless rating criteria are standardized, we may expect that different rating methods will be adopted by different data providers. A different RatingMethodDescriptor provides a description for each of those methods. At the same time, depending on the method adopted, each data provider may give different interpretations to the rating figures. The RatingRangeDescriptor is meant to explain to the data consumer what the rating means. The combination of these two descriptors provides data consumers with the information needed to interpret the raw rating figure. This solution provides great flexibility in defining a customized rating system. Of course, in the happy case where one or both descriptors are standard (this is not the case today), they can be left implicit and only the rating figures for each dimension need to be attached to the data value.

Notice that, for each dimension, some further metadata can be required in order to make the specification of quality ranges, rating ranges and rating
methods as accurate as possible. As an example, when considering an Accuracy evaluation by comparison with an other data source, the data source used for comparison has to be specified. We do not give all the details of such further metadata, as they are strictly related to the specific measurement activity that is carried out by an organization.

5.2.3 The Exchange Phase

The exchange phase is the newly introduced phase of the TDQM_CIS with respect to the traditional TDQM methodology. It is motivated by the need of exploiting opportunities that derive from cooperative scenarios.

In order to realize the cooperation itself, data stored by the different organizations need to be exchanged. Here, the main idea is to have a data integration system, i.e. the Data Quality Broker, that allows to compare same data stored by different organizations, in order to improve their quality at query time. This means that if an organization chooses to join the DaQuinCIS system, it can pose queries to the system with the assurance that the best quality data available in the system will be returned as a result. On the other hand, an organization within the DaQuinCIS system is notified with data available in the system with a better quality than the ones she owns each time she acts as a data provider.

In order to enjoy these benefits, there are two major processes that need to be undertaken by single organizations:

- Assessment of quality values for data willing to export.
- Re-design of some processes for data exchanges in order to exploit the Data Quality Broker functionalities.

The quality assessment has to be performed periodically; the periodicity of the assessment activity depends on data volatility, i.e. frequency of data changes in time, and on the number of new records insertions in a data set.

Re-designing processes in order to improve quality is instead done once. The following example shows how a process can be re-designed in order to exploit the Data Quality Broker functionalities.

Example 5.2.1 Let us suppose that the Alumni Affairs office of the example 5.1.2 wants to check the updateness of the addresses of stored alumni. Let us also suppose that such addresses are stored by three different organizations, namely: City Councils, the Electric Power Company and the Department of Finance. If the Alumni Affairs knows that such addresses are stored by the three organizations, it could ask for them and compare with the ones it stores in order to check the updateness of its addresses. Notice that this
could be a process designed in the Quality Improvement phase of the IP-UML methodology.

Nevertheless, this process is not straightforward to be implemented. Firstly, the Alumni Affairs is supposed to know which are the organizations storing data it needs, and has to directly contact them for data. Secondly, it is not easy to establish among the different copies of addresses provided by the different organizations which is the most update one.

Instead, the described process could be redesigned by introducing the Data Quality Broker, as shown in Figure 5.9. Here, the Alumni Affairs has simply to pose the query asking for addresses to the Data Quality Broker that: (i) discovers which organizations store addresses and asks for them and (ii) once collected addresses with their quality, it can select them on the basis of currency values. □

Besides the principal activities of assessment and process redesign that an organization has to undertake in order to join the DaQuinCIS system, there are also some technological elements that are supposed to be provided, namely:

- Wrappers. Data and quality exported by organizations needs to be wrapped in order to hide technological and modeling heterogeneities.
- Quality Factory. It has to be deployed on each organization with the aim of managing quality values, both for internal assessment purposes and for interfacing the organization with the DaQuinCIS system by evaluating quality values for data.
- Propose Manager. This a module, described in 3.4.2, that has to be implemented inside each organization according to the specific quality improvement policy of the organization (see also Section 5.2.4).

### 5.2.4 The Improvement Phase

In cooperative information system, the main opportunity for improvement comes from data replication: having different copies of the same data stored by different organizations gives the possibility to compare them and improve quality.

Nevertheless, considering the huge amount of data that typically characterizes cooperative systems, it seems not feasible, or at least very difficult, to perform an off-line improvement by comparing such huge data sets tout court. The simple data sets comparison strategy has also the drawback that can’t be performed once, because data vary in time and therefore, a periodicity should be fixed to repeat the comparison process.

In the Data Quality Broker, we implement a strategy for quality improvement that has the following characteristics:
Figure 5.9: An example of process redesign based on the Data Quality Broker usage.
It typically compares only a limited subset of data available in the CIS, which is at worst consisting of the largest set of exchanged data, therefore very rarely consisting of the whole data set of an organization.

- It is an on-line strategy, which means that as query processing goes on, exchanged data are improved. This also means that only data actually exchanged are considered for improvement.

- It respects the autonomy of organizations. Quality feedbacks are sent to organizations that are free of implementing the quality policy they prefer.

We are currently carrying out experiments that compare the effectiveness of the Data Quality Broker improvement strategy with respect to an off-line improvement strategy.

A quality improvement strategy which is complementary to the one implemented by the Data Quality Broker is the one underlying the Quality Notification Service. Indeed, by exploiting the Quality Notification Service, an organization can receive a notification of a quality change on some data to which she subscribed. Then, the organization can directly ask for data of a better quality and update her data set.

## 5.3 The DaQuinCIS Methodology: IP-UML + TDQM_CIS

In the previous sections, we described two methodologies, namely IP-UML and TDQM_CIS, to analyze quality requirements and design quality improvement actions. In this Section, we clarify how the combined use of these proposals can be exploited in an overall methodology for data quality exchange and improvement in cooperative information systems.

An organization that participates to a cooperative system may improve the quality of its own data by two “perspectives”, namely:

- the $\text{ENDO}$ view, i.e. inside-out view, which is the view that a single organization provides of the quality of its own data to other cooperating organizations. The $\text{ENDO}$ view includes many issues, such as all internal quality measurement and improvement actions, maintenance and monitoring activities etc.

- the $\text{EXO}$ view, i.e. outside-in view, which is the view concerned with how internal data quality is influenced from outside. The $\text{EXO}$ view is concerned with all issues that exploit data replication among different systems in order to improve data quality. Therefore, all measurement,
improvement and maintenance methods that can be applied from outside.

These two views are strictly correlated and a complete strategy for data quality improvement must benefit from both. As an example, on one hand, the measurement activities considered by the $\mathcal{E}\mathcal{N}\mathcal{D}\mathcal{O}$ view are necessary for any kind of improvement actions within the $\mathcal{E}\mathcal{X}\mathcal{O}$ view; on the other hand, triggers for internal improvement actions may derive from $\mathcal{E}\mathcal{N}\mathcal{D}\mathcal{O}$ methods.

The IP-UML methodology supports the $\mathcal{E}\mathcal{N}\mathcal{D}\mathcal{O}$ view, whereas the TDQM_CIS methodology supports the $\mathcal{E}\mathcal{N}\mathcal{D}\mathcal{O}$ view.

More specifically, as seen, the IP-UML methodology includes a data analysis phase, in which data internally managed as well as exchanged outside the organizational boundaries are modelled. After this phase, the quality analysis phase allows to specify quality requirements on data: such requirements are useful for driving internal improvement actions, according to the $\mathcal{E}\mathcal{N}\mathcal{D}\mathcal{O}$ view. On the other hand the data and quality analysis phase are necessary for: (i) identifying data to be exported to other organizations; (ii) identifying quality dimensions that are of interest for the organization and that may guide an agreement phase with other organizations on which quality data has to be exported in the cooperative environment; in this sense, the $\mathcal{E}\mathcal{N}\mathcal{D}\mathcal{O}$ view, under which the IP-UML methodology can be placed, provides important inputs to the $\mathcal{E}\mathcal{X}\mathcal{O}$ view.

The TDQM_CIS methodology allows to realize quality improvement from the outside of an organization, according to the $\mathcal{E}\mathcal{X}\mathcal{O}$ view. Indeed, as seen, the improvement phase of such a methodology is based on quality feedback that are sent to organizations in order to internally improve the quality of their data assets.

In Figure 5.10, the interaction among the two methodological processes is graphically shown.

### 5.4 IP-UML: A Case Study

In this section, we describe the main results deriving from the testing of the IP-UML methodology in a real setting provided by the IQA (Improving Quality of Addresses) project, in which all the phases of the process described in Figure 5.1.8 were applied.

After a short description of the IQA setting and objectives, later in this section we provide a description of each phase of IP-UML applied in the context of the project. Notice that, as said, the IP-UML methodology is iterative; therefore, each phase of the methodology was applied more than once. In the following, for the sake of conciseness we will present only the final results of the iteration process.
5.4.1 Improving Quality of Addresses in the Italian Public Administration: the IQA Project

The IQA (Improving Quality of Addresses) project started in 2002 and is supposed to be finished by the end of 2003 [36]. It is jointly managed by the Italian National Institute of Statistics (ISTAT) and the National Authority for IT in the Public Administration (AIPA). The IQA project is also part of the Italian e-Government initiative [9].

The general aim of the project is to make a preliminary analysis for improving the quality of addresses stored in the Italian Public Administration (PA) databases. Addresses are a typology of data that is stored by almost all public administrations; indeed, they have the obvious need of locating people and enterprises in order to communicate with them and thus accomplishing to their specific roles. In Italy, there is the problem that addresses are not standardized; this means that administrations can store addresses as one string, containing values like street denominations, civic numbers etc. or as sets of separate fields; in this latter case no standard reference for such a set of fields is provided.

The specific aims of the IQA project are:

- To propose a possible standard for addresses to be shared by the whole PA.
- To study the causes of data quality problems on addresses.
To improve quality of addresses by a process redesign.

The IP-UML methodology was applied throughout the project in order to achieve such objectives.

### 5.4.2 Data Analysis Phase

A first difficulty we faced in the project was to identify the information product(s). Specifically, a main question we had to solve is related to which type of addresses were in the scope of the project. More specifically, we had to decide if residence addresses of people and addresses locating enterprises should be considered or not. The IP-UML methodology was very useful to such a scope. By considering the flows originating data, we were able to distinguish people and businesses addresses into: (i) pure locating data and (ii) addresses, comprising locating data but with a semantics given by the associated information identifying people or enterprises. In Figure 5.11 an example is provided of pure locating data and addresses related to people’s residences.

We recognized as information product only pure locating data. Instead, addresses were in the scope only because consisting of pure locating data, and thus they are involved only in the notification process, as we will see in Section 5.4.5. The structure of this information product was then identified by distinguishing raw data and component data as shown in a diagram of the Data Analysis model represented in Figure 5.12.
5.4.3 Quality Analysis Phase

As a second step, we needed to precisely define the quality dimensions and the quality requirements on the information product and on its constituents. As the IP was decomposed according to the Data Analysis Model, we specified the quality requirements for raw data and component data. A simplified set of the diagrams composing the Quality Analysis Model is shown in Figures 5.13, 5.14, 5.15 and 5.16.

Specifically, in Figure 5.13, it is specified that the whole IP has to be PureLocData must be accurate.

Figure 5.12: A diagram of the Data Analysis Model showing the composition of the pure locating data IP

Figure 5.13: A diagram of the Quality Analysis Model showing accuracy requirements
5.4. IP-UML: A CASE STUDY

Figure 5.14: A diagram of the Quality Analysis Model showing quality completeness requirements

Figure 5.15: A diagram of the Quality Analysis Model showing currency requirements on the IP
Figure 5.16: A diagram of the Quality Analysis Model showing consistency requirements on the IP
accurate, where accuracy is measured with a boolean metrics; actually, the measurement activity in the IQA project has been carried out according to statistical techniques, and the metric used for accuracy is quite more complex, but details on this are outside the scope of this thesis.

In Figure 5.14, for all the elements composing the IP, it is shown a completeness requirement. Notice that in the figure, only the attribute completeness is shown, i.e. completeness related to the presence or absence of value (that can be also easily represented by a cardinality constraint). Nevertheless, entity completeness was also considered in the project.

In Figure 5.15, it is specified that the whole IP has to be current. Notice that for currency guarantee, the distinction between pure locating data and addresses is particularly important. Specifically, the aim of the IQA project was not principally that of ensuring that a residence address is the most updated one (though this objective was also pursued); instead, the focus was on changes of the components of pure locating data due to territorial changes, such as the transfer of a set of streets from a city council ownership to another.

In Figure 5.16, consistency checks are specified through n-ary associations among some IP-components, namely: (i) Prefix and area Denomination on one hand and (ii) area Denomination and Number on the other hand.

5.4.4 Quality Verification Phase

This phase of the project allowed to identify areas that were quality critical in administrative databases storing pure locating data.

A statistical analysis was implemented, consisting of three sequential phases: (i) the selection of a sample in three major administrative databases (the sizes of the three databases are about 10,000,000, 1,400,000, 4,400,000 records); (ii) the analysis of the sample by using a software for recognition of locating data and (iii) a comparison of the file obtained as output in phase 2 with the input file to phase 1. This method was used for accuracy and completeness evaluation. As for consistency, it was verified by ad-hoc implemented consistency checks. Instead, it was not possible to evaluate currency in an easy way, and this dimension was only considered in the improvement phase. The details of this measurement phase are out the scope of this thesis. Nevertheless, the identification of quality critical areas allowed to model weaknesses in specific processes, according to the aims of the Quality Verification phase.

As an example, one of the results of this analysis is that pure locating data related to Venice have a very poor quality, due to the fact that areas’ denominations in Venice are different from all the areas’ denominations in Italy. Indeed, the notion of street (in Italian “strada”) does not exist in Venice, where instead there is the notion of “calle” which has the same semantics. On the basis of these considerations, it was decided to introduce a quality
check in the data flows concerning the acquisition of locating data related to Venice. The diagram concerning such acquisition, to be implemented by all administrations storing locating data related to Venice, is the one shown in Figure 5.17. In the figure, the case of acquisition due to a change of residence address of a citizen by the residence city council is shown.

5.4.5 Quality Improvement Phase

The quality improvement phase was based on a process re-engineering of inter-administrations processes aimed at improving quality of data. Specifically, the quality requirements specified in the Quality Analysis model were the drivers of the redesign performed in this phase.

We will show in the following an example of process redesign driven by the
currency requirement, shown in the Figure 5.18. Such a requirement imposed to think of an automated notification mechanism to be activated each time an event related to the creation or modification of the elements composing the IP occurred. Notice that, in Italy the responsibility on each element of the identified IP is owned by a different subject; in other words, distinct Data Stewards [80] are identified by law on each raw data or component data associated to the IP. As an example, for the Area element City Councils are the Data Stewards, instead for the CAP’s (the Italian denomination of ZIP codes) the Mail Agency is the official Data Steward.

In the redesign of processes in order to guarantee the currency requirement, the main idea was that Data Stewards of IP’s elements have to notify all the subscribing administrations about an event of creation or modification, occurred on a specific element of the information product, namely City Council, Area, CAP and so on.

Events have been classified into:

- **Denomination Change**, occurring each time an IP raw data element changes its denomination. For instance, if a City Council changes its name, a denomination change event occurs.

- **Birth/Death**, occurring when a new IP element is created or disappears. For instance, if a Province is splitted into two City Councils, one death event related to the Province and two birth events related to City Councils are generated.

- **Land Acquisition/Loosing**, occurring when an IP element acquires or looses some land. For instance if a set of Area is lost by the City Council A and it is acquired by the City Council B, two events of respectively loosing and acquisition types are generated.

In Figure 5.18, the case of a transfer of Areas from a City Council A to a City Council B is described. The Data Steward of the component data Areas is the City Council A, who has to notify the transferred to the City Council B and to a central nationwide index storing all the elements of the information product, that is pure locating data (one of the aims of the project is a preliminary design of this central index for storing locating data). Such index has the role of subsequently notifying the event to all administrations that have subscribed their interest to it.
Figure 5.18: Quality Design Model: A diagram of the Quality Improvement Phase
Chapter 6

Extending the $D^2Q$ Model

6.1 Introduction

The basic idea underlying the $D^2Q$ model, described in Section 3.2, is that the data model of XML is not semantically rich enough to well-model data having a specific semantics, like, for instance, quality data. Therefore, we chose to introduce a new model, namely the $D^2Q$ model, in order to well-model quality data. Starting from this model, we simply use XML as an “implementation” language.

In this chapter, we describe a different idea that we are currently carrying out. Specifically, our aim is to enhance “in general” the semantics of the XML data model in order to assign a different semantics to different hierarchies of data. Indeed, quality data are only an example of data that have a semantics by their own, distinct with respect to data they are associated to.

In this chapter, we define a Multi-Colored Tree (MCT) model [50]; the MCT model enriches the semantics of the XML data model by assigning a different color to each semantically cohesive data hierarchy.

6.2 The Multi-Colored Tree Model

XML (eXtensible Markup Language) is rapidly becoming the de facto standard for exchanging data between applications, and publishing data, on the Web. In part, XML’s popularity is because of its ability to uniformly represent both (i) structured (homogeneous and heterogeneous) data, and (ii) marked-up text, in the same document.

But XML is just syntax. The data model that underlies this XML syntax is a tree-structured model, which consists of nodes, atomic values and sequences [42], and is used by query languages such as XPath [11] and XQuery [17]. This XML data model consists of different kinds of nodes, of which element
nodes are used to capture the nested “structure” of an XML document; and values associated with text and attribute nodes (the leaf nodes of this tree structure) capture its “content”.

The importance of the tree structure in the XML data model can be appreciated from the rich variety of ways in which XPath and XQuery support “navigation” between structurally related nodes in an XML database; this includes use of the parent, child, descendant-or-self, and attribute axes. In contrast, for XML nodes that are related only through values (of the ID/IDREF attributes, or otherwise), one needs to explicitly perform a series of value-based joins, one “edge” at a time, akin to SQL queries over relational databases. The following example is illustrative:

Example 6.2.1 [Deep Trees vs Flat Trees]
Consider a movie database, with movie, movie-genre, movie-award, actor and movie-role element nodes, each of which has a name subelement node. There are many ways of organizing this information in XML, two of which are illustrated below:

Deep-1: The movie-genre nodes are hierarchically organized, with each movie as a child of its primary movie-genre node. Each movie has children movie-award and movie-role nodes, and movie-role nodes have children actor nodes.

Flat-1: The nodes are in a flatter tree structure, and the relationships between movie-genre and movie nodes, between movie-award and movie nodes, and between movie, actor and movie-role nodes, are captured via attributes values. Node have an id attribute, a movie may have a movieAwardIdRefs and a movieGenreIdRefs attributes, and movie and actor nodes have the roleIdRefs attributes.

Now suppose that our query is “Return names of comedy movies that were nominated for an Oscar, in which Bette Davis acted”. In the Deep-1 approach, one can write the following XQuery expression:

```xml
for $m in document("mdb.xml")//movie-genre 
[name = "Comedy"]//movie[
/.//actor/name 
= "Bette Davis"]
where contains($m/movie-award/name, "Oscar")
return <m-name>
{ $m/name }
</m-name>
```

In the Flat-1 approach, one would have to use value-based joins. The XQuery expression would be:

```xml
for $c in document("mdb.xml")//movie-genre 
[name = "Comedy"],
$mg in $c//movie-genre,
```
6.2. THE MULTI-COLORED TREE MODEL

```xml
$m in document("mdb.xml")//movie,
$ma in document("mdb.xml")//movie-award,
$a in document("mdb.xml")//actor
    [name = "Bette Davis"],
$r in document("mdb.xml")//movie-role
where contains($ma/name, "Oscar") and
    $mg/@id = $m/@movieGenreIdRef and
    contains($m/@movieAwardIdRefs, $ma/@id) and
    contains($m/@roleIdRefs, $r/@id) and
    contains($a/@roleIdRefs, $r/@id)
return <m-name>
    {$m/name}
</m-name>
```

Note that the Deep-1 query expression is much simpler than that of Flat-1. The increased complexity of the Flat-1 expression would also (typically) result in a very expensive evaluation, which cannot make an effective use of structural joins developed for the efficient evaluation of XQuery’s path expressions [22, 6].

The improved query specification and evaluation in deeper trees over flatter trees comes at a cost. The deeper representations are un-normalized [7], and the replication of data (e.g., the actor and the movie-award nodes, in the above example) raises the problem of update anomalies (e.g., if one wanted to add a subelement birthDate to each actor). It appears that neither the deep tree approach nor the flat tree approach is ideal both for queries and for updates. What does an XML database designer do in this case?

One possibility might be to use XML views: create deep tree views over the (stored) flat tree data, and let users pose queries against the deep tree views. For example, one could specify Deep-1, in the above example, as an XQuery view over Flat-1. While this would ease query specification, query evaluation over an unmaterialized view would still be expensive. Further, and more importantly, updates would still be problematic. Since updates through XML views can be ambiguous in general (just as for SQL views), users would be forced to specify updates over the flat tree representation, requiring them to be aware of two representations, one for querying and one for updates. This is undesirable as well.

The solution proposed here to effectively address the above-mentioned inadequacies of the conventional XML data model is a novel logical data model for XML data, referred to as multi-colored trees (MCT). Our MCT data model is an evolutionary extension of the XML data model of [17] and, intuitively, permits multiple colored trees to add semantic structure over the individual nodes in the XML data. Individual nodes can have one or more colors, permitting them to be hierarchically related to other nodes in a variety of ways, instead of only in one way. This allows (extended) XQuery expressions to navigate between structurally related nodes, taking color into account, instead of
relying heavily on value-based joins. An (enhanced) XQuery expression can be used to create a new colored tree over a combination of newly created and existing nodes, and an (enhanced) update expression can be used to modify existing data in the MCT data model. The rest of this chapter is organized as follows:

☐ We present the MCT logical data model, consisting of evolutionary extensions to the XML data model, in Section 6.2.4.

☐ We propose extensions to the XQuery query language, for the MCT logical data model, in Section 6.2.5.

Next, in Section 6.2.1, we present an overview of our MCT data model, and highlight its benefits over the conventional XML data model using examples.

6.2.1 Overview of the MCT Model

The W3C has focused considerable recent attention to developing a logical model and query language for XML (see, for example, [42, 17, 93, 16]). The XML data model is an ordered tree of nodes, with atomic values associated with leaf nodes. XQuery expressions allow the concise specification of tree-structured patterns to match against such data. While this data model is adequate for the tasks of encoding relational data (where all relationships are captured using values), or exchanging heterogeneous data between applications, the single tree structure limits the expressivity of modeling.

Using examples, we next highlight the benefits of permitting multiple trees, instead of just a single tree, to add semantic structure over the individual data nodes. Each tree is distinguished from the others by a color, and the resulting logical model is called the multi-colored tree (MCT) data model. We will present a formal development of the MCT data model, which is an evolutionary extension of the XML data model, in subsequent sections.

6.2.2 movie Nodes with Multiple Colors

Consider, again, the movie database from Example 6.2.1. There are several natural hierarchies: movie genres are akin to a topic hierarchy (e.g., comedy and action are sibling genres, and slapstick is a sub-genre of comedy), and the Oscar best-movie awards can be organized into a temporal hierarchy. Individual movies can be naturally classified into each of these hierarchies: a movie can be a child of its most-specific primary movie genre, and, if the movie was nominated for a best-movie Oscar award in a particular year, it can be made a child of that year’s node in the best-movie award hierarchy.
Q1: Return names of comedy movies whose title contains the word Eve.

Q2: Return names of comedy movies that were nominated for an Oscar, whose title contains the word Eve.

Q3: Return names of comedy movies that were nominated for an Oscar, in which Bette Davis acted.

Q4: Return names of actors in movies nominated for an Oscar, with more than 10 votes.

Q5: Return the list of Oscar nominated movies, grouped by the number of votes received.

Figure 6.1: Example queries against movie database

Figure 6.2: Example MCT database

Explicitly modeling such hierarchies in XML allows XQuery expressions to be effectively used for formulating queries like query Q1 (in Figure 6.1), without having to identify the most-specific genre of the movie. While XML allows either of these hierarchies to be modeled, it does not permit a natural modeling of both these hierarchies simultaneously; one of these hierarchical relationships would need to be captured using attribute values, increasing the complexity of the XQuery specification of a query like Q2 (in Figure 6.1).

Our multi-colored tree data model extends the XML data model in permitting both these hierarchies to be first-class semantic hierarchies over the data nodes, simultaneously. Queries like Q2 an be easily expressed in a simple extension of XQuery, that takes color into account in its path expressions. We illustrate an example MCT database in Figure 6.2.

Example MCT Database: As in the XML data model, nodes are given primacy in the MCT data model and each node may have one or more colors. A colored tree consists of all nodes that have a specific color, organized as a
tree. We depict a multi-colored tree database by showing each colored tree separately. The example MCT movie database in Figure 6.2 has trees of three colors: red, green and blue. For the moment, focus on just the red and the green trees. The red tree consists of, among other nodes, the hierarchy of movie-genre nodes, and their associated children name nodes. The green tree consists of, among other nodes, the hierarchy of Oscar movie-award nodes, and their associated children name nodes.

All edges in a colored tree have the same color, depicting the parent-child relationships in that colored tree. A node that has multiple colors is represented in each of its colored trees, e.g., as a green circle with a red outer circle in the green tree, and as a red circle with a green outer circle in the red tree. The same node in different trees is identified by a label which identifies its colors (by the colors’ initials) and a unique node number, e.g., RG012. In the example MCT movie database, a movie node is both red and green (i.e., it participates in both colored hierarchies), if it has been nominated for an Oscar movie-award. A movie node is only red if it has not been nominated for an Oscar movie-award. In the example, the children name nodes of movie nodes have all the same colors as their parents. In addition, movie nodes that are both green and red have green children votes nodes, indicating the number of first-place votes received.

Example Queries: As in XQuery, multi-colored XQuery (MCXQuery) queries are FLWOR (for, let, where, order by, return) expressions, with path expressions replaced by their colored counterparts. An ordinary path expression identifies nodes within a single tree by executing different location steps; each step generates a sequence of nodes and filters the sequence by zero or more predicates. A colored path expression additionally specifies colored labels with each location step, using curly braces, identifying the colored tree(s) to navigate in that location step. Unabbreviated MCXQuery expressions for queries Q1 and Q2 (of Figure 6.1) are given in Figure 6.3.

6.2.3 movie-role Nodes with Multiple Colors

Representing movie nodes in both the (red) movie-genre hierarchy and the (green) Oscar movie-award hierarchy, as in Section 6.2.2, intuitively allows for “entity” nodes that participate in multiple 1 : n relationships to have multiple colors. By allowing for “relationship” nodes to have multiple colors, we can also structurally relate nodes that participate in m : n relationships in a natural fashion.

Consider, again, our movie database in Figure 6.2. All actor nodes and their children name nodes are represented in a (relatively) flat blue hierarchy. Since each movie-role node captures the relationship of an actor with a movie, these nodes (and their children name nodes) can be made both red and
blue: its red parent is the movie node, and its blue parent is the actor node participating in this specific relationship.
Q1: for $m$ in document("mdb.xml")/
    {red}descendant::movie-genre[{red}child::name = "Comedy"]/
    {red}descendant::movie[contains({red}child::name, "Eve")]
    return createColor(black, <m-name> { $m/{red}child::name } </m-name>)

Q2: for $m$ in document("mdb.xml")/
    {red}descendant::movie-genre[{red}child::name = "Comedy"]/
    {red}descendant::movie[contains({red}child::name, "Eve")],
    $m$ in document("mdb.xml")/{green}descendant::movie-award
    [contains({green}child::name, "Oscar")]/{green}descendant::movie
    return createColor(black, <m-name> { $m/{red}child::name } </m-name>)

Q3: for $m$ in document("mdb.xml")/
    {green}descendant::movie-award
    [contains({green}child::name, "Oscar")]/{green}descendant::movie,
    $r$ in document("mdb.xml")/
    {red}descendant::movie-genre[{red}child::name = "Comedy"]/
    {red}descendant::movie[. = $m]/{red}child::movie-role,
    $r$ in document("mdb.xml")/{blue}descendant::actor
    [{blue}child::name = "Bette Davis"]/{blue}child::movie-role
    return createColor(black, <a-name> { $m/{red}child::name } </a-name>)

Q4: for $a$ in document("mdb.xml")/
    {green}descendant::movie-award
    [contains({green}child::name, "Oscar")]/{green}descendant::movie
    [{green}child::votes > 10]/{red}child::movie-role/{blue}parent::actor
    return createColor(black, <a-name> { $a/{blue}child::name } </a-name>)

Q5: createColor(black, <byvotes> { for $v$ in distinct-values(document("mdb.xml")/
    {green}descendant::votes)
    order by $v$
    return <award-byvotes>
    { for $m$ in document("mdb.xml")/
    {green}descendant::movie[{green}child::votes = $v]
    return $m
    } <votes> { $v } </votes>
    </award-byvotes> } </byvotes>)

Figure 6.3: Example MCXQuery queries
6.2. THE MULTI-COLORED TREE MODEL

Note that, to demonstrate the flexibility of our MCT data model, we have chosen (arbitrarily) not to let movie-role nodes be green, even if the movie was nominated for an Oscar movie-award.

Having modeled actor nodes and movie-role nodes, we can now use multi-colored XQuery to concisely express query Q3, as in Figure 6.3.

MCT is a logical data model, structuring data values using multiple hierarchies, and the bulk of this section is devoted to MCT (Section 6.2.4), and its related query and update languages (Section 6.2.5).

6.2.4 The MCT Logical Data Model

In this section, we formally develop the MCT logical data model, which we motivated and illustrated using examples in the previous section. MCT is an evolutionary extension of the XML data model of [42], and, hence, is presented as such. As we shall see in the next section, this evolutionary approach allows us to build on query and update languages proposed for XML to obtain manipulation languages for MCT databases.

Multi-Colored Trees

The multi-colored trees (MCT) data model enhances the XML data model (see Section 3.3.1 for details) in two significant ways:

- Each node has an additional property, referred to as a color, and nodes can have one or more colors from a finite set of colors $\mathcal{C}$.
- A database consists of one or more colored trees $T_c, c \in \mathcal{C}$, where each node in $T_c$ has color $c$ (as one of its colors).

More formally, we have:

**Definition 6.2.1** [Colored tree] Let $\mathcal{N}$ be a finite set of nodes of the seven kinds defined by the XML data model, and $\mathcal{C}$ be a finite set of colors. A colored tree $T_c = (\mathcal{N}_c, \mathcal{E}_c), c \in \mathcal{C}$, where (i) The set of nodes $\mathcal{N}_c \subseteq \mathcal{N}$; (ii) The set of edges $\mathcal{E}_c \subseteq \mathcal{N}_c \times \mathcal{N}_c \times \mathcal{N}_c$ defines an ordered, rooted tree, satisfying the tree relationships imposed by the XML data model between the different kinds of nodes, with a triple $\langle n_p, n_l, n \rangle$ specifying that node $n$ has $n_p$ as its parent and $n_l$ as its left sibling.1 □

Essentially, a single colored tree is just like an XML tree. Allowing for multiple colored trees permits richer semantic structure to be added over the individual nodes in the database.

---

1We use the convention $\langle n_p, n_p, n \rangle$ to identify node $n$ with parent $n_p$, and no left sibling.
Definition 6.2.2 [MCT database] A multi-colored tree (MCT) is defined as a triple \((\mathcal{N}, \mathcal{C}, \{T_c\})\), where (i) each \(T_c, c \in \mathcal{C}\), is a colored tree; (ii) \(\mathcal{N} = \bigcup_c \mathcal{N}_c\); and (iii) each attribute, text and namespace node \(n_1\) associated with an element node \(n_2\) in any of the colored trees has all the colors of \(n_2\), and has \(n_2\) as its parent node in each of its colored trees.

An MCT is said to be an MCT database if the root of each of its colored trees is the same document node (which, hence, has all colors in \(\mathcal{C}\)), else it is an MCT database fragment.

Intuitively, in an MCT database (fragment), a node belongs to exactly one rooted colored tree, for each of its colors. This is similar to the XML data model, where a node can belong to exactly one rooted tree. Unlike an XML database, however, there is no global document order of nodes in an MCT database: each colored tree defines its own local order of nodes, obtained by a pre-order, left-to-right traversal of the nodes in the colored tree.

As an example, consider the example MCT database fragment of Figure 6.2. It consists of three colored trees, with the colors red, green and blue. All nodes shown are element nodes (with the text nodes implicitly present as children of element nodes depicted with values in parentheses). For example, nodes \(RG012\) and \(RG015\) are both red and green, and nodes \(RB117\) and \(RB125\) are both red and blue.

As in the XML data model, we use abstract accessors and constructors to explain the data model. These are discussed in the next two sections.

Node accessors

In the XML data model [42], ten accessors are defined for all seven kinds of nodes. Four of these accessors, namely, \(dm:\text{parent}\), \(dm:\text{string-value}\), \(dm:\text{typed-value}\), and \(dm:\text{children}\), would need to be extended to take a color into account. Their signatures are given in Figure 6.4. If the node on which these accessors are called does not have the color that is passed as an
6.2. THE MULTI-COLORED TREE MODEL

argument to the accessor, an empty sequence is returned. Otherwise the node and the accessor are said to be color compatible, and the desired result is returned from the appropriate colored tree.

The other six accessors defined in the XML data model, namely, \texttt{dm:base-uri}, \texttt{dm:node-kind}, \texttt{dm:node-name}, \texttt{dm:type}, \texttt{dm:attributes}, and \texttt{dm:namespaces}, are not influenced by the color of the node, and continue to have the same signature and meaning as in the XML data model.

In addition, a new accessor needs to be defined on all node kinds to determine the colors of a given node:

\[
\texttt{dm:colors($n \text{ as Node}) as xs:string+}
\]

Node Constructors

In the XML data model, each node kind defines its constructors, which always return a new node with unique identity. This is feasible since the nodes can be constructed iteratively, in a “bottom-up” fashion in the XML tree. In our MCT data model, it is not always possible to construct a node only after all its children in each of its colors have been constructed, e.g., \texttt{element node} \(n_1\) may be a child of \texttt{element node} \(n_2\) in one color, but a parent in a different color. To effectively permit the construction of multi-colored trees, we define two different types of constructors for each node kind.

- \textit{First-color} node constructors are like constructors in the XML data model, except that they are extended to take a color into account, and return a new node with unique identity.

- \textit{Next-color} node constructors take a previously constructed node, and add a color and the tree relationships in that color; the same node is returned.

Example constructor signatures for the \texttt{element} node are depicted in Figure 6.5. Note that the signature of the next-color constructor is somewhat smaller than that of the first-color constructor, since one does not need to repeat some of its properties, and its \texttt{attribute} and \texttt{namespace} nodes.

```
dm:element-node
($qname as xs:QName, $nsnodes as NamespaceNode*, $attrnodes as AttributeNode*, $children as Node*, $type as xs:QName, $color as xs:string) as ElementNode
dm:element-node
($self as ElementNode, $children as Node*, $color as xs:string) as ElementNode
```

Figure 6.5: Modified and new node constructors
The MCT logical data model defines allowable syntactic structures. The semantics of the database are captured by its schema. The XML schema language proposed by the W3C deals with both structure [93] and datatypes [16]. Formally extending XML schema to multi-colored trees is an interesting direction of future work.

6.2.5 Data Manipulation Languages

We now formally develop the MCXQuery logical query language, which we motivated and illustrated using examples in the previous section. The MCT data model, being an evolutionary extension of the XML data model, allows us to naturally build our logical query language as an extension to XQuery.

MCXQuery Path Expressions

An XQuery path expression can be used to locate nodes in tree-structured XML data (see Section 3.3.1 for details).

An example of an XQuery path expression related to our example is:

```
document("mdb.xml")/child::movie-genre
descendant::movie-genre[name = "Comedy"]
$c/descendant-or-self::movie-genre
```

In the MCT logical data model, a node may have multiple colors, in which case it would belong to multiple colored trees. Hence, an axis and a node test specification (e.g., `parent::node()`) does not suffice to uniquely identify the navigation to be performed in a single step, from a context node. For example, in the MCT database of Figure 6.2, the `movie` node RG012 has two parent nodes: a `movie-genre` node in the red tree, and a `movie-award` node in the green tree. However, since a node belongs to exactly one rooted colored tree, for each of its colors, augmenting the specification of a step by a color would serve to provide the necessary disambiguation.

We achieve this by enclosing the color specification in curly braces, preceding the axis specification in the step expression, e.g., `{red}descendant::movie` and `{blue}child::movie-role`. The extensions to the relevant productions in the grammar of XQuery are shown in Figure 6.6. In general, different steps in an MCXQuery path expression may have different color specifications, and
the resulting navigation over the MCT database can be quite sophisticated. The result of evaluating an MCXQuery path expression is, as before, a sequence of items, all of which have the same color, as determined by the color specification of the final step in the path expression. The order of items in the result sequence is determined by their local order in the corresponding colored tree.

Figure 6.3 presents a few illustrative path expressions in MCXQuery, with each step augmented by a color specification. Query Q4, in particular, illustrates the use of different color specifications in different steps of the path expression.

**MCXQuery Constructor Expressions**

XQuery provides constructor expressions that can create XML tree structures within a query, based on the notion of constructors for the different node kinds in the XML data model.

When the name of the element to be constructed is a constant, the element constructor is based on standard XML notation. Enclosed expressions, delimited by curly braces (to distinguish them from literal text),\(^2\) can be used inside constructors to compute the content of the constructed node, and also its attributes.\(^3\) Enclosed expressions are evaluated and replaced by their value (which may be any sequence of items). For example, the return clauses of the Deep-1 and Flat-1 queries in the introduction have constructor expressions with enclosed expressions.

Since every tree in the MCT logical data model is a colored tree, XQuery constructor expressions are suitable for the creation of new colored trees in an MCT database as well. One such constructor expression could be used for the creation of each colored tree, and an MCT database/fragment could be created using multiple constructor expressions. One key issue needs to be resolved, however. The result of an element constructor in XQuery is always a new element node, with its own identity; all the attribute and descendant nodes of the new element node are also new nodes with their own identities, even though they may be copies of existing nodes.

Always creating a new node is inappropriate for constructor expressions in MCXQuery, since such a node would have a different identity from existing nodes in the MCT database, limiting the capability of MCXQuery constructor expressions in creating MCT databases/fragments, where nodes belong to

---

\(^2\)Note that the use of curly braces for enclosed expressions does not conflict with their use to specify color when navigating steps in path expressions.

\(^3\)A special form of constructor called a computed constructor can be used in XQuery to create an element node or attribute node with a computed name or to create a document node or a text node. These can also be extended, in an analogous fashion, for MCXQuery.
multiple colored trees. To effectively permit the construction of multi-colored
trees, MCXQuery constructor expressions need the ability to *reuse* existing
nodes and their descendants, in addition to being able to create element nodes
with new identities. This is achieved as follows.

- When an enclosed expression is evaluated, its value (a sequence of items)
  *retains* the identities of nodes in the sequence, instead of creating copies
  by default. This is similar to the behavior of MCXQuery path expres-
sions.

- To create node copies, MCXQuery provides a function named *createCopy*.
The *createCopy* function takes any sequence of items as its argument,
and returns copies of the items in the sequence, in the same order.

For example, the result of evaluating the enclosed expression in the *return*
clauses of queries Q1, Q2 and Q3 in Figure 6.3 would contain the node with
identity RG015, since identities are preserved when the enclosed expression is
evaluated. If, however, the *return* clause contained the constructor expres-
sion:

```xml
<m-name>
  createCopy({ $m/ {red} child::name })
</m-name>
```

the result would contain a new node, with a different identity.

To associate a color with the result of a constructor expression, MCXQuery
provides a function named *createColor*. This function takes two arguments:
a color literal as its first argument, and any sequence of items as its second
argument. It *adds* the specified color to the set of colors associated with each
node in its second argument.

Finally, we address an interesting issue that arises if node identities are
retained when evaluating enclosed expressions in a constructed expression,
especially when this result is colored. Since a node can be present at most
once in any colored tree, the result of any constructed expression *should not*
have a node (with a given identity) occur at more than one position in the
colored tree. Such a situation can arise, as the following constructed expression
illustrates:

```xml
createColor(black, <dupl-problem>
  <m1> { $m/ {red} child::name } </m1>
  <m2> { $m/ {red} child::name } </m2>
</dupl-problem>)
```

In this case, the expression raises a dynamic error. Note that such a
situation doesn’t arise if the *createCopy* function is appropriately used.
6.2. THE MULTI-COLORED TREE MODEL

**FLWOR Expressions**

XQuery’s FLWOR (for, let, where, order by, return) expressions support iteration and binding of variables to intermediate results, and are especially useful for computing joins between multiple documents and for restructuring data (see Section 3.3.1 for details).

The enhancements made to path expressions and constructor expressions, discussed above, allow FLWOR expressions to be directly usable as MCXQuery expressions. Q1–Q4 in Figure 6.3 present four example MCXQuery expressions, corresponding to the queries posed in Figure 6.1 against the movie database. Note that the path expressions used in the for clauses use color specifications at each step, and the constructor expressions used in the return clauses are colored black, by the use of the createColor function.

Finally, we present an example depicting how an MCXQuery expression can be used to add a new colored tree, consisting of new nodes and previously existing nodes, to an MCT database. Consider query Q5 from Figure 6.1. The MCXQuery expression is shown in Figure 6.3. The result of evaluating this expression against the MCT database of Figure 6.2 is shown in Figure 6.7. Notice that movie nodes now have three colors: red (because of their participation in the movie-genre hierarchy), green (because of their participation in the movie-award hierarchy), and black. All other nodes, including the newly created votes nodes in the result, are only black. Note that the result is a tree since each movie in the green movie-award hierarchy has only one child element named votes.

**The Update Language**

There is as yet no standard for specifying updates to XML data. In [92], the authors propose an extension to XQuery (using for, let, where and update clauses) to perform updates to XML documents. Essentially, the for, let and where clauses return tuples of bindings, one of which is the target of the sequence of update operations in the update clause. The update operations
include node deletion, node renaming, node insertion, node replacement, and the ability to nest these update operations in an update expression.

It is easy to see that the MCXQuery extensions to XQuery path expressions and constructor expressions, described previously, can be used in a straightforward manner in conjunction with the update extensions of [92], to unambiguously update an MCT database. Each of the update operations can be performed on existing colored trees, once the tuple of bindings is returned. Note that update operations implicitly add existing colors to new nodes, or to existing nodes. Creating new colored trees is done via extensions to the constructor expressions in MCXQuery.

6.2.6 The MCT Model to Represent Quality Values

An example of how the MCT model can be used to associate data quality values to data is shown in Figure 6.8. In order to represent different data quality dimensions, we introduce a hierarchy of a specific color for each dimension; each colored hierarchy corresponding to a quality dimension contains values of the metric scale used for that dimension. In the figure, accuracy is represented by a red hierarchy and currency is represented by a green hierarchy. The chosen scale for the example in the figure is \{at least low, low, at least medium, medium, high\}, but any suitable scale can be used for measuring dimensions. Besides quality data, also data are represented by a colored hierarchy. In the figure, data are represented by the blue hierarchy; actually, the shown data a subtree of the \(D^2Q\) schema instance example shown in Figure 3.6. Only accuracy values are associated to \texttt{FiscalCode} and \texttt{Name} nodes, while both accuracy and currency values are associated to the \texttt{Address} node.

On the basis of this example, we envision to use the multi-colored tree model in place of the \(D^2Q\) model in the \texttt{DaQuinCIS} system. Specifically, in order to associate quality values to data, we would like to exploit the MCT’s feature of not restricting to have a single hierarchy over data to be represented in XML.
Figure 6.8: An MCT instance to represent data quality
Chapter 7

Concluding Remarks

In this thesis, a complete approach for exchanging and improving data quality in cooperative information systems is defined and implemented, namely the DaQuinCIS approach. Many research issues have been investigated for the realization of the DaQuinCIS approach.

A summary contribution to the data quality definition problem has been provided, whereas no data quality definition finds a common agreement in current literature.

A data model has been defined for representing data and quality data, namely the $D^2Q$ model. We are currently working on an extension of the model based on quality-colored trees, according to what introduced in Chapter 6. Such an extension could be also supported by visual querying tools that further potentiate the “navigability” of the model.

We have defined a data integration system that allows to query data and to access quality values associated to data, i.e., the Data Quality Broker. The system needs to be extended in different directions by removing the underlying simplifying hypothesis. Furthermore, some optimizations can be performed on the Data Quality Broker basic behavior. Firstly, if renouncing the quality improvement function of the Data Quality Broker, the set of sources to be queried can be chosen on the basis of past performances of sources in providing data with high quality values. Statistics can be stored on the quality levels of data provided by each organizations, that can be used to optimally answer queries. Secondly, system level dimensions, such as availability and response time could be taken into account for both optimizing performances of querying and making the user able to specify trade-offs between quality of data and system characteristics. For instance, a user may want to obtain lower quality data, as long as they are provided in a shorter time.

We have also incorporated an improvement feature at query processing time in the Data Quality Broker. We’d like to study how such an on-line
improvement could be combined with an off-line improvement, consisting of record matching procedures periodically run. To this aim, the Record Matcher module used within the Data Quality Broker can also be used as a stand-alone module performing off-line improvement.
Bibliography


167


