Data Quality Maintenance in Data Integration Systems

Adriana Marotta

PhD Thesis

Thesis Advisor: Professor Raul Ruggia

Universidad de la República – Uruguay

March, 2008

ABSTRACT

A Data Integration System (DIS) is an information system that integrates data from a set of heterogeneous and autonomous information sources and provides it to users. Quality in these systems consists of various factors that are measured in data. Some of the usually considered ones are *completeness*, *accuracy*, *accessibility*, *freshness*, *availability*. In a DIS, quality factors are associated to the sources, to the extracted and transformed information, and to the information provided by the DIS to the user. At the same time, the user has the possibility of posing quality requirements associated to his data requirements. DIS Quality is considered as better, the nearer it is to the user quality requirements.

DIS quality depends on data sources quality, on data transformations and on quality required by users. Therefore, DIS quality is a property that varies in function of the variations of these three other properties.

The general goal of this thesis is to provide mechanisms for maintaining DIS quality at a level that satisfies the user quality requirements, minimizing the modifications to the system that are generated by quality changes.

The proposal of this thesis allows constructing and maintaining a DIS that is tolerant to quality changes. This means that the DIS is constructed taking into account previsions of quality behavior, such that if changes occur according to these previsions the system is not affected at all by them. These previsions are provided by models of quality behavior of DIS data, which must be maintained up to date. With this strategy, the DIS is affected only when quality behavior models change, instead of being affected each time there is a quality variation in the system.

The thesis has a probabilistic approach, which allows modeling the behavior of the quality factors at the sources and at the DIS, allows the users to state flexible quality requirements (using probabilities), and provides tools, such as certainty, mathematical expectation, etc., that help to decide which quality changes are relevant to the DIS quality. The probabilistic models are monitored in order to detect source quality changes, strategy that allows detecting changes on quality behavior and not only punctual quality changes. We propose to monitor also other DIS properties that affect its quality, and for each of these changes decide if they affect the behavior of DIS quality, taking into account DIS quality models.

Finally, the probabilistic approach is also applied at the moment of determining actions to take in order to improve DIS quality. For the interpretation of DIS situation we propose to use statistics, which include, in particular, the history of the quality models.

CONTENT

CHAPTER 1. INTRODUCTION

1. Context

A Data Integration System (DIS) is an information system that integrates data from a set of heterogeneous and autonomous information sources and provides it to users. It is an integration system that follows the GAV (global as view, [Chawathe-04]) approach, and basically consists of a set of data sources, a transformation process that is applied to data extracted from sources, and a global data view, which is the data access provided to users. We consider the transformation process as a workflow in which the workflow activities perform the different tasks that extract, integrate and transform data such that it satisfies end-users information needs. We consider that the DIS may have different degrees of data materialization, falling in one of three categories: (i) Virtual, where all the information extraction, integration and transformation is done at the moment of the user query, (ii) Materialized, where there exists an integrated schema whose information is materialized, and (iii) Hybrid, where there is an integrated schema where some of its parts are materialized and other ones are virtual.

The user, who retrieves information from the DIS, is usually far from its generation and ignores how and when it was published at the sources as well as what transformations it suffered. This situation worsens as sources are more external and out of user control. Therefore, the user may not feel fully confident about the retrieved information. This is always undesired and becomes critical when the user is going to make decisions based on the obtained information.

For this reason, the management of data quality in this kind of systems becomes an essential issue. Quality in these systems consists of various factors that may be somehow measured in the data. Some of the usually considered ones are *completeness*, *accuracy*, *accessibility*, *freshness*, *availability*. In a DIS, quality factors are associated to the sources, to the extracted and transformed information, and to the information provided by the DIS to the user. At the same time, the user has the possibility of posing quality requirements associated to his data requirements. Considering that we have, on one hand the quality offered by the DIS, and on the other hand, the quality required by the user, we assume the quality of the DIS as better, the nearer it is to the user quality requirements (this consideration is shared by other authors, e.g. [Strong+97]).

Quality factors are measured at the data sources, and then they can be calculated for the data provided by the DIS to the user. Quality at data sources may be measured by the owners of the sources or by the DIS administrator, once he has extracted it or basing in certain information that allows him to deduce the quality values. We consider quality factors whose state can be represented by numerical values. For example, we associate to the quality factor *freshness*, numerical values that represent the oldness of the data since certain moment.

The values of the quality factors provided by the DIS can be evaluated, taking as input data sources quality values. There are many works that focus on this problem, such as [Naumann+99][Mecella+03][Peralta+04].

Considering the concepts expressed above, it is clear that DIS quality depends on data sources quality, on data transformations and on quality required by users. Therefore, DIS quality is a property that varies in function of the variations of these three other properties.

This work is closely related to a previous one, the thesis work of Verónika Peralta [Peralta-06], which is situated in the same context. It mainly addresses the problems of quality factors definitions and quality evaluation in DIS. Our work is based on many of its results and can be seen as a continuation of it. It is for this reason that many references to the mentioned work of Peralta can be found throughout this thesis.

2. Motivation

We use an example for supporting the motivation of our work.

Consider a system that integrates information from several hospitals of a country, which is used by an epidemiology department of the government in order to make decisions concerning the country inhabitants. The system is called *HealthDIS*. The freshness of the information obtained from the system is very important, since the decisions may be quite different according, for example, to the number of cases of a disease appeared the last day. One of the users, called *Peter*, stated for certain query about diseases cases, *query1*, a required freshness ≤ 10 hs. The DIS was designed so that all users' freshness requirements were satisfied. However, the freshness of the information obtained by Peter changes very frequently, since different conditions of *HealthDIS* are continuously changing. At a certain moment one of the sources starts being updated with a lower frequency than before. Peter is probably making wrong decisions because he is using data that is not fresh enough. Which is even worse, he trusts on this data and thinks it has a freshness that it does not really have. Analogous situations are generated with respect to accuracy of the obtained information. Figure 1.1 shows the DIS.

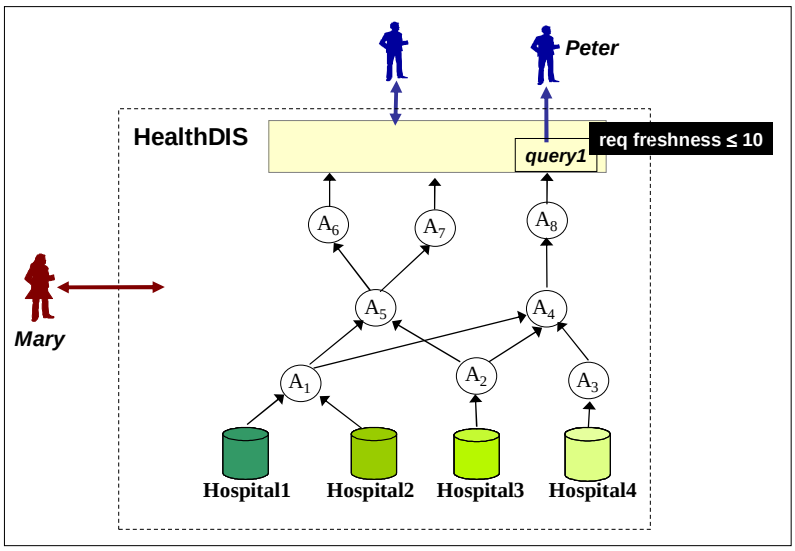


Figure 1.1: Example: HealthDIS

The DIS must have as an objective the maintenance of the satisfaction of the user quality requirements. On the other hand, it is absolutely necessary that the DIS guarantee Peter that he will always be aware of the quality of the obtained information. It cannot happen that the user thinks that the DIS is satisfying his quality requirements when this is not true. To achieve this, the system must be prepared to endure or detect and manage the changes that affect its quality.

The administrator of *HealthDIS*, called *Mary*, is in charge of the maintenance of DIS quality. The most serious problem she has is related to the system restructuring that may be necessary each time quality changes. Sometimes, the system looses quality and it must be restructured in order to recover its previous quality. This may imply, for example, to eliminate a data source from the system, to change some data transformation process, etc., which are in general very expensive system modifications. The DIS administrator needs to minimize the frequency of system restructuring.

It would be a great solution for the previously described situation to have a DIS that is **tolerant to quality changes**. This means that the DIS is constructed taking into account previsions of **quality behavior**, such that if changes occur according to these previsions the system is not affected at all by them. These previsions could be provided by models of quality behavior of DIS data, which should be maintained up to date. With this strategy, the DIS would be affected only when quality behavior models change, instead of being affected each time there is a quality variation in the system.

In summary, it is necessary to have models of the behavior of DIS quality. With this powerful tool, we would be able to construct and maintain a DIS that is tolerant to quality changes. Through the detection of DIS-quality behavior changes, instead of only punctual quality changes, the impact of quality changes on the DIS is minimized.

3. Problem Statement

A preliminary analysis of the problem of DIS quality changes can be found in Appendix I, where we characterize the phenomenon and we compare it with a well-known similar problem.

The problem we address is basically how to maintain an acceptable level of quality in a DIS. Normally, there are diverse and frequent changes in these systems that may affect their quality, leading us to apply corrective modifications to it. It is important to avoid unnecessary modifications, since they may be deep ones, such as changes on the design, elimination or substitution of the participant sources, etc. Therefore, it is necessary to correctly determine when an occurred change on the DIS merits this kind of actions. It is desirable to have a control over the DIS that allows us to monitor it, know the behavior of its quality, detect when we should act for repairing its quality and repair it when it is necessary.

We should have as much knowledge as possible, about the DIS; its past, present and probable future, such that we can have a clear idea of its situation when we have to modify it to improve its quality.

We distinguish three basic problems to address:

1- How can we know the quality behavior of the DIS?

Quality is measured in source data and we are able to calculate from it the quality of information provided to the user. It is clearly possible to calculate the quality provided by the DIS at a certain moment, measuring quality of the used source data and calculating the resulting quality, or directly measuring quality in the resulting data. However, it would be more useful to know how DIS quality behaves in general, having an estimation of the quality that may be obtained in the queries to the DIS. We want to have a model of this behavior. A model of DIS quality behavior will depend on the models of sources quality behaviors. All these models must be calculated and also maintained.

2- How can we detect a change that has occurred on the DIS, affecting its quality behavior and taking it to an unacceptable state?

Many different changes may occur in the DIS. These changes may have different degrees of incidence on DIS quality. We must define techniques for identifying when an occurred change affected the DIS quality to a degree that it turns unacceptable. For this we must take into account the user quality requirements. It is very important to detect which are the relevant changes in order to minimize the actions for compensating changes on the DIS.

3- How can we repair DIS quality after a change?

There are many modifications that can be applied to the DIS in order to improve its quality. It is necessary to study how the different components relate to each-other and how their different characteristics affect the quality factors. In addition, when we choose some modifications, they may not be the most suitable ones, i.e. the most convenient in a cost/benefit relation, for the given DIS situation. Therefore, the most important issue in this problem is to correctly diagnose the DIS situation and to determine the best actions to be applied for repairing its quality. In the resolution of this problem it may be taken into account the change that generated the dissatisfaction and all the information that is maintained about the DIS and its quality.

4. Goal of the thesis

The general goal of this thesis is to provide mechanisms for maintaining DIS quality at a level that satisfies the quality requirements of end-users, minimizing the modifications to the system that are generated by the quality changes.

For achieving this general goal it is necessary to reach some particular or specific sub-goals:

- Provide techniques for modeling DIS quality behavior and maintaining these models.
- Propose strategies for monitoring DIS quality.
- Propose a mechanism for detecting relevant DIS quality changes.
- Propose a mechanism for repairing DIS quality

In addition, there is another sub-goal, which is the experimentation of an important part of the proposed techniques in a real case. This is useful to show the feasibility and applicability of the proposal.

5. Approach of the thesis

The thesis has a probabilistic approach, which allows modeling the behavior of the quality factors at the sources and at the DIS, allows the users to state flexible quality requirements (using probabilities), and provides tools, such as certainty, mathematical expectation, etc., that help to decide which quality changes are relevant to the DIS quality.

We propose to monitor the probabilistic models in order to detect source quality changes, strategy that allows detecting changes on quality behavior and not only punctual quality changes. We propose to monitor also other DIS properties that affect its quality, and for each of these changes decide if they affect the behavior of DIS quality, taking into account DIS quality models.

Finally, the probabilistic approach is also applied at the moment of determining actions to take for improving DIS quality. To interpret DIS situation we propose to use statistics, which include, in particular, the history of the quality models.

In conclusion, the general approach used throughout the thesis is the probabilistic vision of the quality at the system. As secondary approaches we use the management of events for detecting changes and rules for managing them.

We think that this is a proactive approach, since it is based on actions we can perform before the occurrence of a change: a) calculating how source and DIS quality can vary without failing to satisfy the quality requirements of the DIS, and (b) building probabilistic models, which allow

predicting quality behavior. This prediction allows having a DIS that is tolerant to quality changes, since it is maintained so that it supports the predicted changes.

The main advantage of this approach is that it minimizes the impact of quality changes on the DIS, avoiding a lot of useless work in its maintenance.

It is difficult to give behavior patterns or rules, and detailed solutions, applicable to the majority of the quality factors. Therefore, to study more deeply the problem of changes it is necessary to start considering one factor at a time, since each factor has a particular behavior, and affects the system differently. In our work we focus on two quality factors: freshness and accuracy. We have chosen these factors because they have very different characteristics, such as the form of propagation in the system or the parameters that affect them.

We propose the existence of a system that coexists with the DIS, which we call Quality Management System, where our proposed mechanisms are carried out.

6. Contributions

The main contributions of this thesis are the following:

- Techniques for modeling quality behavior in a DIS
  We propose to build and maintain probabilistic models of the quality factors at the sources and DIS. We provide techniques for modeling quality of the sources for freshness and accuracy, which apply to different scenarios. We also provide techniques for modeling quality of the DIS for freshness and accuracy factors.
- A mechanism for detecting relevant quality changes in a DIS
  We provide a mechanism for detecting changes of DIS quality from events that notify certain changes in different DIS elements. After processing the events and evaluating the effects of the changes it notifies only the relevant changes. The main advantage of the mechanism is that it filters a lot of changes that are not relevant and selects only the changes that deserve a treatment, which are changes on quality behavior that generate the dissatisfaction of user quality requirements.
- A mechanism for analyzing DIS situation and finding the most suitable actions for recovering quality
  We provide a mechanism that from the occurred relevant changes analyzes the situation of the DIS, basing on statistical information maintained in the management system, deduce an interpretation of the situation and then determines a ranked list of actions for recovering DIS quality. The mechanism basically consists of sets of rules, which can be extended adding new rules, new interpretations and new actions. The provided interpretations, actions and rules show the usefulness of the mechanism.

7. Document Organization

The present document is organized as follows.

Chapter 2 presents the existing knowledge about quality management, specially quality changes management.

Chapter 3 presents an overview of our proposal. It also presents some basic concepts that are used throughout the proposal.

Chapter 4 presents the techniques for building the sources quality models and the DIS quality models.

Chapter 5 presents the proposal for change detection; first a taxonomy of changes and then the mechanism for detecting relevant quality changes at the DIS.

Chapter 6 presents the proposal for quality recovery. It first presents an analysis of the possible and effective modifications that can be applied to the DIS after a quality change, and then it presents the proposed mechanism for deducing which actions are the most suitable for each situation of the DIS.

Chapter 7 presents a description of the experimentation that was carried out in order to show the applicability of the proposal.

Chapter 8 presents the conclusion of the thesis.

CHAPTER 2. EXISTING KNOWLEDGE

1. Introduction

In this chapter we present an overview of the existing knowledge in the areas that are related and relevant to our work.

With respect to Data Quality area, we briefly present some of the most relevant works about the general field, data quality dimensions and measurement. We analyze more in depth some works about data quality evaluation in Data Integration Systems, since our work is strongly based on them, and finally we emphasize and focus on analyzing the works about data quality improvement, since this is the sub-area that is closest to the one addressed in the present thesis. Therefore, we mostly concentrate on analyzing data quality improvement proposals, whose approaches we found closest to ours.

The separation in the sub-topics *quality dimensions*, *measurement*, *evaluation in DIS* and *improvement*, perhaps is not so realistic or natural, in fact many proposals involve several of them and they strongly inter-relate. Nevertheless we opted to organize the works in this way because it allows us to concentrate in which we are most interested and, on the other hand, it is useful for the positioning of our work.

In addition to quality-related research works, we comment some existing works about schema evolution and probability techniques applications. Our intention is to show works that have influence in our approach and solutions.

In Section 2 we present the works in the area of Data Quality, in Section 3 we comment some works about source schema evolution problem, in Section 4 we comment some applications of probability techniques to data management, in Section 5 we position our work with respect to the previously presented ones, and finally, in Section 6 we present the summary of the chapter.

2. Data Quality

Data Quality is a wide research area, which involves many different aspects and problems, and also important research challenges. On the other hand, it has an enormous relevance for industry due to its great impact on information systems usefulness in all application domains. A great amount of work about data quality can be found in the literature, mostly generated in the last decade. An interesting analysis of the evolution and current state of the field is presented in [Neely-05]. In this paper, the author combines the five principles for product and service quality defined by J. M. Juran in [Juran-88] and the framework for analysis of data quality research presented by Wang et al. in [Wang+95-a]. Based on this combination he analyzes in which problems existing work most focuses and which ones have not had much attention. Her main conclusions are the following. There is significant research regarding the production and distribution of information, in particular related to data warehousing. Also the problem of quality dimensions has been well researched, while more work still needs to be done with regards to measurement or metrics. There is very little work related to economic resources and operation and assurance costs. The author claims that it is time to move beyond the definition of data quality dimensions and determine how these dimensions define the quality of data in terms of the user. She also states that more research in improving the analysis and design of information systems to include quality constructs is needed. Finally she poses the need of supporting frameworks with empirical data.

Another interesting analysis of data quality research area is the presented in the paper written by Scannapieco *et al.*, [Scannapieco+05-a]. Here, the authors, after insisting in the importance of

addressing data quality problems and improving quality of data in many contexts, mention a definition of data quality that is referenced in many other papers ([Neely-05], [Wang-98]): *fitness for use*. Then they focus on the multidimensional characteristic of data quality, as researchers have traditionally done. They precisely define the dimensions: *accuracy*, *completeness*, *currency* and *consistency*. They say that this core set of dimensions is shared by most proposals in the literature, although the research community is still debating the exact meaning of each dimension, and studying which is the best way to define data quality. Various general sets of data quality dimensions have been proposed including the mentioned core set and further dimensions. Finally they comment that for specific application domains, it may be appropriate to have more specific sets of dimensions.

With respect to quality dimensions, we highlight the following works. In [Wang+95-b] the authors motivate the need for data quality dimensions definition, as the need for tagging data with quality indicators which are characteristics of the data and its manufacturing process. They introduce a large set of data quality dimensions and a hierarchy of them. In [Strong+97], based on the concept that high-quality data is data that is fit for use by data consumers, the authors define certain data quality dimensions and categories. Finally, quality dimensions and classifications of them into categories are presented in [Lee+02], where the authors provide a table summarizing the academics' view of information quality (quality properties defined by different researchers), and another table summarizing the practitioner's view (quality properties defined by specialists within organizations, consultants, vendors of products).

In our research group, we have also worked on the study and definition of data quality dimensions. We presented, in a joint work with R. Ruggia, [Marotta+03], a study of a wide set of quality dimensions and we propose a correspondence between user-viewpoint and system-viewpoint quality dimensions. As an example of definition of quality dimensions specific to an application domain, there is the work presented in [Etcheverry+07], which focus on defining the appropriate quality dimensions for the biological domain, specifically for microarray databases.

There are works that concentrate in very few quality properties, such as [Peralta-06], where a very deep study of freshness and accuracy quality dimensions is presented, and [Bright+02] and [Theodoratos+99], where freshness and its impact in the system design is studied in depth.

Oriented to Data Warehouse (DW) environments, many data quality works can be found. In [Ballou+99] the authors bring the data quality problem to the DW area. In [Jarke+97] the authors present a set of quality factors, grouped in categories that may influence a DW system. They discuss several relationships between quality parameters and design/operational aspects of a DW. The work presented in [Jeusfeld+98] gives a formal meta-model for representing quality goal formulation and quality measurement in a DW. Examples of specialization and instantiation of the model are presented. In [Calero+01] the authors focus on multidimensional models' quality. They present a set of quality metrics for a DW "star" design, and a formal validation process that is applied to them.

Addressing the problem of quality measurement, many works can also be found. We highlight, for example, the one in [Pipino+02], where data quality assessment is presented as depending on *subjective perceptions* and *objective measurements*. Subjective assessments reflect the needs and experiences of stakeholders (collectors, custodians and consumers of data). In many cases different stakeholders have different assessments for the same quality dimensions. Objective assessments, which involve metrics for the data set in question, can be task-independent (the metrics do not take into account the context of the application) or task-dependent (the metrics are developed in specific application contexts). In practice, the authors propose performing subjective and objective assessments, comparing the results and identifying discrepancies in order to take actions for improvement. On the other hand they propose three functional forms for developing objective

metrics: *Simple Ratio, Min or Max Operation*, and *Weighted Average*. The simple ratio is the number of undesirable outcomes divided by total outcomes subtracted from 1. The min or max operation is used when the dimension measurement requires the aggregation of multiple data quality indicators. The weighted average is an alternative to the min or max operation, useful when there is a good understanding of the importance of each variable to the overall evaluation of a dimension. The authors present a set of data quality dimensions and the metric for each one, applying these three forms.

We also highlight the work in [Naumann+00], where the authors present a classification for information quality criteria according to the possible sources of the criteria scores. They classify them in three classes: subject-criteria, object-criteria and process-criteria. The first is when the scores can only be determined by individual users (e.g., understandability). The second one is when the scores can be determined exactly by analysis of information (e.g., completeness). The third one is when the scores are determined by the process of querying (e.g., response time). They analyze the characteristics of each kind of measurement and they propose general methods for each one. They also comment the confidence of the measurement in each case. They emphasize the importance of having a detailed description of how and when the assessments should take place, remarking the convenience of repeating the assessment regularly.

Other important works include proposals for measuring certain quality dimensions, such as accuracy and currency, [Shankaranarayan+03] [Ballou+95][Ballou+03], also taking into account weights that contemplate context characteristics. In [Even+05] they have an approach where measurement is oriented to the content and its applicability for business use, instead of basing upon impartial characteristics of the data (assuming standards for the measurement).

In addition, there are several works that measure other quality aspects of data, such as [Calero+01], which focuses on multidimensional models' quality, and [Moraes+07], which measures quality in data integration schemas.

2.1 Data Quality Evaluation in DIS

In systems whose information is obtained from multiple sources, integrated and eventually transformed in different manners, to know the quality of the information provided to the user is not a trivial problem. Even in the case where the quality of sources is perfectly known and informed by the sources owners, the quality provided by the DIS to the user must be calculated. We call data quality evaluation to this calculation, which in some cases is actually an estimation.

In the following paragraphs we comment some of the works that address this topic, which we found and selected from the literature, due to their influence or relation with our approach.

In [Naumann+99] the authors present a quality model in a heterogeneous information system, which allows calculating quality values for the possible plans of a query. They propagate quality factors through the query plans in order to deduce the quality of them. They consider a plan as a binary tree with QCAs (query correspondence assertions between the sources and the mediator) as leaves and join-operators as inner nodes. They propose to use a function *Merge* to obtain the factor value of a relation that is the result of a join, from the factor values of the participating relations. In particular, they define accuracy quality factor as the percentage of objects without data errors such as misspellings, out-of-range values, etc., and the Merge function they propose for accuracy is the product of the accuracy values of the joining relations.

In [Ballou+06] the authors propose techniques for estimating accuracy in tables that are the result of combinations of base tables. On one hand, they propose a method (Reference-Table Procedure) that estimates the accuracy of the result of any operation, doing a comparison between the resulting

table and a reference "correct" one. Samples of the base tables are used. The reference table is a sample obtained from the corrected base tables samples. Then the sample obtained from the original base tables samples, is compared to the reference table, and from this comparison the resulting accuracy is determined. On the other hand, they propose some formulas for estimating accuracy from the accuracy of the samples of the base tables. The combinations they consider are the basic operators of relational algebra. For the cases of Selection and Projection operations the estimated accuracy is the same as the estimated accuracy of the input table. In the case of Union operation, they give the following formula: $P = (n_1*P_1 + n_2*P_2) / (n_1 + n_2)$, where n_1 and n_2 are the number of tuples of the input tables and P_1 and P_2 are the estimated accuracy values of each of them. In the cases of Projection and Union the respective formulas are useful when no duplicates are generated in the result; for both operations, in the cases of existence of duplicates (which are analyzed separately) they propose the use of the Reference-Table Procedure. For Cartesian Product and Join by foreign key operations, the proposed formula is the following: $P = P_1 * P_2$, where P_1 and P_2 are the estimated accuracy values of the input tables. The estimation of accuracy for Join over non-foreign key attributes is very complex. They analyze various cases for showing the complexity of the problem. They also propose for each estimation the calculation of the confidence interval, and they finally present techniques for obtaining appropriate samples.

The work presented in [Shankaranarayan+03] involves many aspects of the problem of quality management, but, due to our interest here, now we only pay attention to their proposal for evaluating accuracy factor. They work basing on a model for the processing of an information product, called IPMAP, which we better describe in next section. In this model there are different constructs, such as processing blocks. A processing block combines data units to create a different data unit. The accuracy of the output data unit is dependent on the processing performed. The proposed formula is for a generic process that combines multiple data elements to create an output, not taking into account the type of processing performed and ignoring the error (in accuracy) that might be introduced by the process itself. They propose to do a weighted average of the input accuracy values: $\Sigma_{i=1,n} (a_i * A_i) / \Sigma_{i=1,n} (A_i)$, where a_i is a weight provided by the decision-maker for the input data unit i and A_i is the accuracy value of the input data unit i.

In [Pon+05] the authors propose a method for ranking several sources basing on their accuracy, giving the possibility to DIS of selecting the more accurate source data and knowing its accuracy. In this context there is a set of data sources, from where the system extracts the same data, therefore each source data can be compared to each other. The proposal for evaluating the accuracy of a source consists of a formula that considers the following three issues: (1) a probability of the source of being absolutely accurate, (2) the previous estimated accuracy (at the previous time unit), and (3) the agreement with the other data sources. They are based on two main ideas. The first one is that data sources that have been accurate in the past are also likely to be accurate in the future. The second one is that if a data source agrees with an accurate data source, it should also be accurate, and if a data source agrees with an inaccurate data source, it should also be inaccurate.

In [Peralta-06] the author defines the *Quality Evaluation Framework*, which is intended to be a flexible context which allows specializing evaluation algorithms in order to take into account the characteristics of specific application scenarios. For example, in a DIS that materializes data, the data freshness evaluation method should take into account the delays introduced by data refreshment, while in a virtual DIS such delays are not applicable. The framework models data sources, data targets and the DIS processes. DIS processes include the tasks for extracting, transforming and integrating data and conveying it to users. The DIS is modeled as a workflow process that includes these tasks, and they define the concept of *Quality Graph* for representing and managing it. The graph is directed and acyclic. Its nodes are of three types: activity nodes, source nodes and target nodes, and its edges are of two types: control edges and data edges, which in general coincide. The quality graph is adorned with property labels that allow estimating the quality

of the data that can be produced by the DIS, for example, the time an activity needs for executing or a descriptor stating if an activity materializes data or not. Quality evaluation is performed by evaluation algorithms that calculate the quality values for the graph.

She proposes a basic algorithm for evaluating data freshness, which takes into account the freshness of source data but also the amount of time needed for executing all the activities and the delays that may exist among their executions. The algorithm propagates freshness actual values traversing the quality graph and calculating the freshness of the data outgoing each node. The calculation is done as follows:

- For an activity node A with one predecessor P, the freshness of data outgoing A is calculated adding the freshness of data produced by P, the inter-process delay between P and A, and the processing cost of A.
- If an activity A has several predecessors, the freshness of data coming from each predecessor (plus the corresponding inter-process delay) is combined and added to the processing cost of activity A. The typical combination function computes the maximum of the input values, but other user-specific functions may be considered.

Figure 2.1 shows an example of freshness evaluation through the described algorithm (*Afreshness* means actual freshness).

Then the work presents a general algorithm for evaluating freshness and studies different scenarios and ways to instantiate the algorithm.

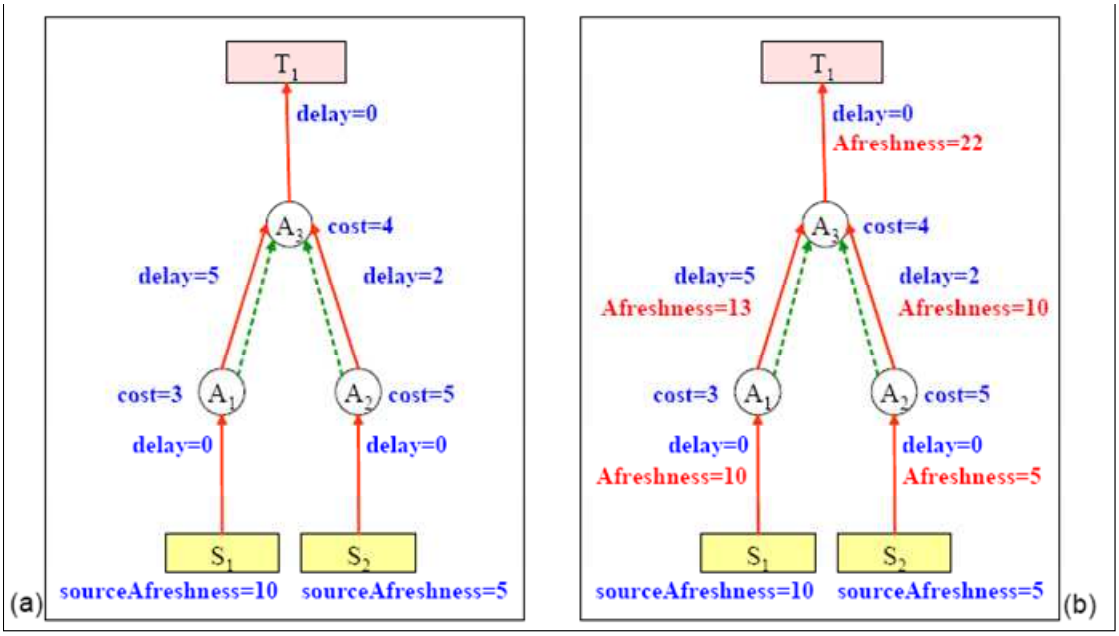


Figure 2.1: Example of Freshness Evaluation (from [Peralta06])

The author also proposes a method for evaluating accuracy quality factor. The method considers that accuracy is not necessarily homogeneous over a whole source table, however partitions of a source table can be defined where homogeneity can be assumed. Therefore, her proposal for accuracy evaluation consists of three steps: (1) Partitioning source relations according to accuracy homogeneity, (2) Rewriting user queries in terms of partitions, and (3) Estimating data accuracy of query results. The estimation of data accuracy for each partition is done assuming accuracy

homogeneity, which simplifies the problem. At the end of the process, accuracy values of the resulting partitions are aggregated for obtaining one value of accuracy for the target data obtained. We do not comment in depth the process of evaluation proposed.

In Table 2.1 we present a summary of the previously commented characteristics of the works. It is important to note that we only make reference to the characteristics we are interested to investigate in this section, which are not necessarily among the most important contributions of each considered work.

Work	Quality Evaluation Proposal
[Naumann+99]	- Merge function for join operations. - Merge for accuracy factor: product of input accuracies
[Ballou+06]	- Evaluation based on Samples of the source data. - Reference-table Procedure for any operation. - Estimation of accuracy: - Selection and Projection (no duplicates): it is maintained. - Union (no duplicates): weighted average of the input accuracies (weight: number of tuples). - Cartesian Product and Join (by fk): product of input accuracies
[Shankaranarayan+03]	- For accuracy, formula for a generic process that combines multiple input data: weighted average of input accuracies, where weights are given by decision-maker.
[Pon+05]	- Evaluates accuracy considering: - Probability of the source of being accurate - Previous estimated accuracy - Agreement with other sources
[Peralta-06]	- Evaluates freshness considering input freshness, inter-process delay and activity cost. If many inputs to the activity, uses Maximum. - Evaluates accuracy generating partitions that have homogeneous accuracy.

Table 2.1: Quality evaluation proposals

We have also done some specific works in the area of data quality evaluation. In particular, in [Marotta+06] we propose evaluation techniques for accuracy, freshness and availability factors in the context of ROLAP systems. We provide a set of formulas that allow estimating or calculating the values of these factors, for the result of any multidimensional operation of a predefined basic set. These multidimensional operations were: *dice*, *projection*, *drill-across*, *roll-up*, *change-base*, and *union*. Basing on their expressions in function of the basic relational algebra operators, a calculation is derived for each quality factor. The calculation also takes into account the type of the elements that participate in each operation (in ROLAP systems we distinguish, e.g. a measure attribute from a dimension attribute). For the calculations of the factors in the basic algebra operations we based ourselves on the different formulas proposed in some of the previously commented works.

2.2 Data Quality Improvement

Data quality improvement is a very vast topic. Its scope is very wide since it involves many different aspects of data management and can be treated with many different approaches. Much work oriented to this problem can be found in the literature, mostly generated in the last few years.

Under the "umbrella" of data quality improvement we identify three main different kinds of works: (i) those which point to improving practices relative to data, inside an organization, (ii) those which are directed to systems that manage data coming from multiple and heterogeneous sources, each one providing their own data quality, and (iii) those which propose specific quality improvement techniques oriented to solve certain data quality problems. The proposals of the different groups are complementary; in fact they frequently appear as portions of other of them. For example, proposals from group (ii) often include or need to be complemented with proposals from (iii).

2.2.1 Practices-Oriented Improvement

The works we group here, focus on managing data quality inside an organization, i.e. they are oriented to data generated in the context of an organization. Therefore, they do not consider external data sources that have a given quality; they consider data whose generation may be under their control. Due to this reason they mainly orient their solutions to the improvement of data related practices.

The most relevant research effort we found in this direction is TDQM (Total Data Quality Management) [TDQM]. TDQM is a program developed at MIT that aims to establish a theoretical foundation in data quality management field and, from this work, to devise practical methods for business and industry to improve data quality. One of the main components of this research is the improvement component, which involves redesigning business practices and implementing new technologies in order to significantly improve the quality of corporate information. They address various methods for improving data quality, which are grouped into four categories: (i) business redesign, (ii) data quality motivation, (iii) use of new technologies, and (iv) data interpretation technology. All of them are intended to improve the different mechanisms of generation of data in an organization, in order to minimize poor data quality.

The publications we highlight in this project are, on one hand, [Wang-98], [Shankaranarayan+00] and [Lee+02], which present the project general approach and proposals, and on the other hand, [Madnick+01], [Madnick+04] and [Wang+05], which present a more specific approach oriented to *corporate householding*.

In [Wang-98] the authors claim that to increase productivity, organizations must manage information as they manage products. They refer to an information manufacturing system as a system that produces information products. The TDQM methodology proposes the continuous iteration of four tasks: Define, Measure, Analyze, Improve, in order to proactively improve the quality of the information product, continuously. In particular, Analysis phase involves investigating the root causes for current IQ problems. To achieve improvement, information manufacturers and suppliers need to expand their knowledge about how and why the consumers use information, while information consumers need to understand how information is produced and maintained.

In [Shankaranarayan+00] the IP-MAP is presented; a modeling method for representing the manufacture process of an IP (information product). This method provides several possibilities to the IP manager, which are very useful for IQ (information quality) management. In particular, it allows him to identify the critical phases and bottlenecks in the manufacturing of an IP, that affect its quality, and to identify ownership of the processes at each of these phases also helping in implementing quality-at-source. It allows IP managers to understand the manufacture process of the IP and to measure its quality at the various stages of the process. In addition, the source of a data quality problem in an IP can be traced in its manufacture process. The modeling constructs in the IP-MAP consist of construct blocks. Among all the block types, they define the Data Quality block, which is used to represent the checks for data quality on data items, and the Data Correction Block, which is for applying corrective actions when quality problems are identified.

In [Lee+02] AIMQ is presented, which is a methodology oriented to solve the problem of information quality assessment and improvement in organizations. The authors focus on giving assessment methods and techniques that allow an organization to compare its information quality to others' and to benchmarks, and also to analyze it across different roles with respect to information (consumers and managers). They assert that this support is essential for an organization to face the problem of data quality improvement.

The methodology is based on three components. The first one, PSP/IQ Model, is a model of what IQ means to information consumers and managers. The second one, IQA Instrument, is a questionnaire for measuring IQ along the dimensions important for information consumers and managers. The third one, IQ Gap Analysis Techniques, consists of two analysis techniques for interpreting the assessments.

The PSP/IQ model classifies the dimensions into four quadrants: sound, dependable, useful, and usable information. The IQA instrument measures IQ for each of the IQ dimensions, and generates measures for the four quadrants. The IQ Gap Analysis techniques are used to benchmark the quality of an organization's IQ and to identify IQ problem areas and focus improvement activities. It consists of two techniques: IQ Benchmark Gaps and IQ Role Gaps. IQ Benchmark Gaps assesses an organization's information quality against a benchmark, which corresponds to a best-practice organization. IQ Role Gaps compare the IQ assessments from IS professionals and information consumers respondents. For each quadrant, it shows the degree of agreement about the level of IQ that there is between information consumers and IS professionals. It is used for determining whether differences between roles are a source of a benchmark gap.

The articles relative to corporate householding are [Madnick+01], [Madnick+04] and [Wang+05]. These articles propose a data quality improvement approach oriented to solve a specific kind of problems. The addressed problems refer to data from corporate household, concept that is deeply explained in [Madnick+01], and basically are how to obtain correct data from a corporation when this data comes from different and diverse components of the corporation. They show that corporate inter-relations must be studied and modeled and rules must be stated in order to correctly process the obtained data. They also focus on understanding the relations between the data and the context where it is being queried, in order to assure that the obtained results are the expected ones. They state the following three categories of corporate householding problems: entity identification, entity aggregation and transparency of inter-entity relationships. Entity identification concerns solving the problem of multiple representations of the same entity (very frequent in corporate entities). Entity aggregation refers to how data must be aggregated, considering the structure of the corporation, which may be very complex, and the context of the query. Transparency of inter-entity relationships refers to the existence of relationships between corporate entities that involve multiple layers, which must also be taken into account at aggregation time.

We also classify in this group some work from M. Scanappieco, presented in [Scannapieco+02] and [Scannapieco+05-b]. In this work a UML profile for data quality is proposed with the aim of supporting quality improvement inside an organization. All the proposal is based on the use of UML and IP-MAP framework [Shankaranarayan+00], which graphically describes the process by which the information product is manufactured.

The profile consists of three models: (i) the data analysis model, (ii) the quality analysis model and (iii) the quality design model. (i) is a model that represents different classes of data: raw data, semi-processed information and information product, and a class named quality data that generalizes the others. (ii) models the quality requirements, which includes a classification of quality dimensions, and relates these requirements to the data represented in (i). (iii) represents the processes that manage data. These models are a support for analysis that allows detecting potential quality

problems and, after having checked the non-conformance to quality requirements, introducing quality improvement actions.

In addition, this work presents a methodology for data quality improvement, which consists in three phases: data analysis, quality analysis, and quality improvement design, each of which leave as result the corresponding models previously presented. With respect to the processes applied to improve quality they propose the use of quality improvement patterns, so that solutions and experiences are reused. These patterns mainly consist of the description of a problem and the description of a solution.

Finally, in [Caballero+04] the authors propose a framework for modeling, assessing and improving quality of information in an organization. They emphasize the importance of an integrative framework for assessing and improving information quality that is based on knowledge about the company and the "information manufacturing processes" (viewing information as a product). The proposal defines two main components: (i) an information quality management model based on maturity staged levels, and (ii) an assessment and improvement methodology. (i) defines five information quality management maturity levels: Initial, Definition, Integration, Quantitative Management and Optimizing. The levels are ordered by taking into account information quality goals and their relative importance. (ii) The main actions proposed by the methodology are the measurement of the state of maturity level and the definition of a plan for improvements.

2.2.2 Data-Processing-Oriented Improvement

The proposals of this group are oriented to systems that manage data coming from multiple and heterogeneous external sources, each one providing their own data quality. In these systems the quality of source data is given and the integration system must deal with this fact, having the possibilities, for example, of selecting sources, negotiating with sources, post-processing source data, combining source data in the most convenient way according to its quality needs, etc.

All these works have as an objective the improvement of data quality, some of them address data quality maintenance, and some others concentrate on quality change management.

We analyze the following four groups of works, which we found in the literature as conference papers, journal articles and thesis works: (1) works from C. Cappiello *et al.*, (2) DaQuinCIS project, (3) work from V. Peralta, and (4) works from P. Bugajski *et al.*.

Proposals of C. Cappiello *et al.* address data quality improvement through quality monitoring, quality problems detection, and quality recovery. Their works are commented in the following paragraphs.

In [Cappiello+06-a] the Hybrid Information Quality Management (HIQM) methodology is proposed. It provides a methodological approach for run time error detection and correction management. Detection of errors in the run-time phase of the process is supported and suitable improvement actions are enforced. The methodology consists of eight phases, from which the ones related to quality improvement are: *Analysis & Monitoring*, *Improvement Functions* and *Strategy Correction.* The methodology basically proposes to measure data quality, compare it to data quality requirements and if they are not satisfied find out the causes and identify the suitable improvement actions. In the improvement actions, both data-oriented and process-oriented techniques are considered. However, the most emphasized contribution of the paper is the Warning generation and management of the methodology. This management enables to monitor data and processes in real-time. The different modules of the warning management interact, detecting three different data quality faults: discrepancies between internal and external data (e.g. web sources), inconsistencies

between data quality values and quality requirements, and anomalies in data management in the system. In addition, data quality problems are identified from feedbacks received from different actors. Finally there is a real-time recovery module that applies recovery actions based on rules for problem solving.

In [Cappiello+06-b] the authors say that in the context of systems using Web Services, organizations should continuously check the quality of the owned and exchanged data. They propose the same methodology as previously. The methodology is proposed as a support for self-healing environments, allowing solving run time data quality problems. An analysis of the business process must be done in the design phase, identifying critical points in the business tasks that may worsen data quality. In these points, information quality is continuously monitored, i.e. it is measured each time information passes through the points. They work with accuracy, completeness and timeliness dimensions. They classify faults along two categories: value mismatch and missing data. The possible causes mentioned in the paper for these errors are: typos, delays in update operations between two databases that contain the same data, value unavailability. For error recovery they propose two types of methods: data oriented and process oriented. The former includes data cleaning, which can be: manually comparing to real world, comparing to other databases or correcting pre-defined errors. The latter refers to modifications to the process structure.

In her thesis work [Cappiello-05], Cappiello devotes a chapter to data quality management, which, according to her, must include algorithms for measuring data quality and automatic techniques for the improvement of data when their quality decreases below acceptable values. This chapter includes an analysis of quality monitoring and evaluation, distinguishing between two possible approaches: (a) on line evaluation, where data quality is computed against each user request, and (b) off line evaluation, where quality values are pre-computed and stored on a quality repository. (a) involves an additional cost to compute the result of the query. On the other hand, it guarantees that the quality metadata are up-to-date. For requests of large data sets, it could be more appropriate to consider the results provided by the off line evaluation process. (b) is independent of the execution of a particular query. Evaluations are done periodically and sometimes they are specifically invoked. It reduces the response time, but it may provide out-of-date information about the quality of data. The stored quality metadata do not take into account all the changes performed in the time interval between two periodic assessments. The author considers as a critical issue the definition of this time interval.

This chapter also includes the proposal of the *Quality Factory*, which supports the data quality assessment and improvement methodology. The methodology is based on rules, which allow the interaction between the Assessment module and the Monitoring module of the Quality Factory, and allow the Monitoring module to evaluate whether quality improvement actions are needed. Rules are also used to determine when quality must be evaluated. For improvement actions they discuss the two possibilities: actions based on data-oriented or process-oriented techniques. The former are appropriate when data are not modified frequently, as they are expensive and have short-term effects. The latter prevents future errors with a long-term effect. Process oriented improvement aims to identify the causes of data errors and eliminate them permanently. Improvement actions change data access and update activities through process analysis and redesign. As a support, they propose to maintain historical information where they maintain the modifications they apply and how they affect data quality.

In [Cappiello-04] the authors present a table summarizing the characteristics of on line and off line evaluations (see Table 2.2). In this paper they also present an evaluation of the methodology through its implementation. We remark two of the aspects they mention. The first one is that the assessment algorithms, when performed on-line, generate a response time that is excessively high, since in the application case there are large amounts of data and users require low response times. They say that in this respect, the off-line evaluation is preferable, although it provides quality values

valid at the time of the last assessment. The second aspect concerns the improvement phase. They say that a fundamental result is that it is not necessary to undertake improvement actions every time the system identifies a quality problem, and that it is important to estimate the returns from an improvement action through a cost-benefit analysis.

	On-line Evaluation	Off-line Evaluation
Evaluation input	Query submission	Periodically Upon specific events Upon request of data quality administrator
Data granularity	Small amount of data (query results)	Large amount of data (usually, entire databases)
Improvement methods	Search for an alternative source	Data or process-oriented improvement methods

Table 2.2: Comparison between on-line and off-line evaluation approaches. (From [Cappiello-04])**.**

The DaQuinCIS project [DaQuinCIS] has as main objective the definition of an integrated framework that includes: (i) an integrated methodology for data quality enhancement in cooperative systems, and (ii) a distributed architecture supporting data quality monitoring and improvement. Some of their proposals are presented in [Scannapieco+04] and [Mecella+03]. These papers propose an architecture for managing data quality in cooperative information systems (CIS). The architecture aims to avoid dissemination of low qualified data through the CIS, by providing a support for data quality diffusion and improvement.

CIS are characterized by high data replication; different copies of the same data are stored by different organizations. They propose an approach for data quality improvement that takes profit of this characteristic, comparing the different copies, selecting the most appropriate one or reconciling them obtaining an improved one. The proposed architecture fits in the TDQM_CIS methodological cycle [Bertolazzi+01], which has five phases: Definition, Measurement, Exchange, Analysis and Improvement. The architecture consists on different modules that support the mentioned phases, but the paper focus on two of these modules: the Data Quality Broker and the Quality Notification Service, which mainly support the Exchange and Improvement phases.

The Data Quality Broker performs the *quality brokering function*, which consists on posing a data request with quality requirements over the other cooperating entities, and the *quality improvement function*, which selects the best-quality value and proposes it to the organizations that can choose to discard their data and to adopt higher quality ones. The quality improvement feature consists of notifying organizations with low quality data about higher quality data that are available through the CIS.

The Quality Notification Service is a publish/subscribe engine that allows subscriptions for users to be notified on changes of the quality of data [Marchetti+03]. When a change in quality happens, an event is published by the Quality Notification Service i.e., all the users which have a consistent subscription receive a notification. An interesting aspect is that the Quality Notification Service can be used in the CIS to control the quality of critical data, keeping track of its quality changes and being always aware when quality degrades under a certain threshold.

The paper deeply presents all the previous concepts an also covers other aspects such as a model for data and quality data exported by cooperating organizations.

The thesis work of V. Peralta [Peralta-06] treats quality improvement problem for freshness quality factor in DIS (Data Integration Systems). Its approach is based on providing, on one hand, facilities for the analysis of freshness problems, and on the other hand, some possible freshness improvement strategies. As a support for these strategies, they provide a set of basic actions over the transformation process of the DIS (called improvement actions) and combinations of them for achieving the different freshness improvement strategies.

The facilities for freshness problems analysis consists of two main techniques. The first one is the top-down analysis of data freshness, that is, to first analyze data freshness in a high-level quality graph[1] and analyze more detailed quality graphs when further details are needed. The second one is the calculation of a critical path on the quality graph for a target node (node that represents the information delivery to the user), which represents the bottleneck for data freshness. In the following we comment more in detail these two proposals.

To achieve the possibility of top-down analysis the author defines a hierarchy of representations of the DIS processes, which is composed by quality graphs at different abstraction levels. The root represents the whole DIS. High-level activities abstract high-level tasks while lower-level activities show the processing details of the tasks. Ascending in the hierarchy implies abstracting task behaviors while descending in the hierarchy implies decomposing an activity in more detailed sub-tasks. All the mechanisms for browsing among the different levels' graphs and for calculating the properties of a graph from the previous graph in the hierarchy, are specified. The properties of the quality graph are the ones that allow evaluating freshness in it, so freshness can be evaluated at any graph of the hierarchy.

Given a quality graph, a path for a target node is a *critical path* if starts at a source node and ends at the target node, and its freshness is equal to the freshness of the data delivered in the target node (target-node's freshness). The freshness of a path (path freshness) is defined as the sum of source-node's freshness, the processing costs of the nodes in the path and the inter-process delays among the nodes. They define these concepts in the context of the assumption that the combination function (calculation of freshness value in an activity node) returns the maximum of input freshness values, and they prove that for such function the critical path always exists. They also prove that critical paths are those that have the greatest path freshness. We transcribe the example they propose for an intuitive idea of the concept of critical path:

> **Example 2.1:** (from [Peralta06])
>
> Consider the quality graph of Figure 2.2. The freshness of data produced by activity A6 (delivered to target T2) can be calculated adding the source data actual freshness of source S1 (0), plus interprocess delays (0,0,10,20) and processing costs (30,60,30,5) in the path from S1 passing by activities A1,A3,A5 and A6, i.e. 0 + (0,0,10,20) + (30,60,30,5) = 155. So, this path is a critical path for T2.

[1] Graph proposed in the work for evaluating quality in the DIS, which is based on the data transformation process performed by the DIS.

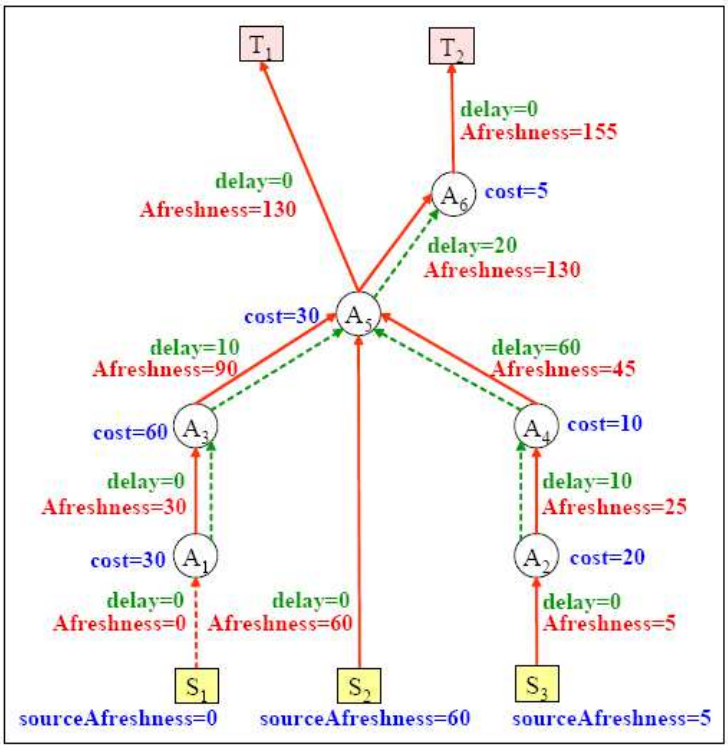


Figure 2.2: Example of Critical Path (from [Peralta06])

◊

The critical path identification is useful because it is the area of the quality graph that should be modified in order to improve data target freshness.

As said before, this work also proposes some improvement strategies and basic actions for achieving them. The strategies they propose are the following: reducing processing costs, reducing synchronization delays, reducing source data freshness, and augmenting target required freshness. Different possibilities for the application of these strategies are analyzed. They propose a set of elementary actions that modify the quality graph topology or properties, for example "AddNode", "AddProperty", "RemoveNode". Over these actions they propose some macros that are useful for applying the mentioned strategies. For example, "ReplaceSubGraph", which may be used for replacing a set of activities by a set of more performing components or replacing a source and its wrapper by new ones providing fresher data.

The works of Bugajski *et al.* focus on data quality changes detection and analysis of quality problems.

In [Bugajski+05-a] the authors propose a framework that basically allows: modeling data baselines (off line analysis), monitoring data and comparing it to baselines (on line), and analyzing relationships between data in order to find causes of quality problems. Considered data is event based, for example, payment transactions.

They claim that instead of focusing on quality factors (accuracy, completeness, consistency, etc.) we should focus on deviations from baselines for measures derived from those. The latter are certain data that is significant for the quality factor and that constitutes persistent features associated

with the business events. They propose then to focus on changes. In addition, they propose to think distributions, not values. For this objective, they build baseline models, which are baseline distributions. They compare the observed model (distribution) to the baseline model (distribution). To build baselines, they continuously observe streams of data and keep persistent state information, i.e. features that maintain across data.

Data are divided into homogeneous segments and for each segment a baseline model is constructed. The construction of baselines and the monitoring of the data streams allow detecting statistically significant changes in data, through detecting deviations from baselines. For example, in a payments card transaction case, they verify if the number of declined transactions is different than the baseline.

Root cause analysis is intended to provide an identification of conditions that are correlated with variables of business interest. Investigations are undertaken involving domain experts to explore these relationships between data.

The paper approach is mainly to present the proposed framework through its application to real cases in payments card domain and highway traffic domain. In [Bugajski+05-b] they go more in depth in showing the construction of the baseline models.

In the paper [Bugajski+06] the authors present more in depth the construction of the baseline distributions for a particular application case: Visa. The problem they approach is the generation of a very big quantity of different baselines that are necessary for this case. They propose the construction of a data cube that is split in different cells, for each of which a baseline model is constructed. They discuss the problems generated by the enormous size of data managed; the determination of the cells, the construction of thousands of baselines, and the management of many alerts that must be studied by the domain experts. They present two concrete case studies of the method application to Visa context.

Finally, in [Curry+07] the authors show the application of the proposal in Visa case study, remarking that they have demonstrated through this case study that change detection using data cubes of baseline models is an effective framework for computing changes on large, complex data sets.

2.2.3 Specific Quality Improvement Techniques

We classify here works that propose concrete techniques to be applied to data, in order to improve its quality. We found a very large quantity of works for data cleaning, which refers to improving accuracy-related factors in data, and on the other hand, a work that proposes techniques for improving data freshness.

A great amount of work is proposed for data cleaning, which consists of techniques for improving factors of accuracy dimension in data. A wide variety of errors are identified and classified, and techniques for the different kinds of errors are proposed. In this work, we do not present an exhaustive or in-depth analysis of data cleaning proposals, we only intend to briefly comment some relevant bibliography.

Some papers that show an interesting analysis of data cleaning are [Rahm+00], [Oliveira+04], [Oliveira+05], [Müller+03], [Quass-99]. In [Rahm+00] the authors provide a classification of data quality problems in data sources differentiating between single- and multi-source and between schema- and instance-level problems, giving an overview of the main existing solution approaches. They also give an overview of commercial data cleaning tools. In [Oliveira+05] the authors present a taxonomy of data quality problems, organizing them by granularity levels of occurrence. They formally define a large number of quality problems, classifying them according to the data

granularity to which they are applied. In [Oliveira+04], the same authors present a classification of the quality problems and the existing proposes for their solutions (data cleaning techniques). In [Müller+03] a survey of data cleansing problems, approaches, and methods, is proposed.

Generalizing, with respect to the classification of errors, some of the errors identified and classified are: (a) illegal values, such as values outside domain range, misspellings and value entered in the wrong field, (b) illegal format, such as unidentifiable abbreviations, no standard format (e.g. in the date) and no standard units, (c) multiple values entered in one attribute, (d) consistency problems, such as integrity constraints violation, and (e) incorrect values with respect to real world.

Some important projects in data cleaning are the ones presented in [Galhardas+00], [Galhardas+01], which present the AJAX tool, [Vassiliadis+01], with ARKTOS tool, [Raman+01], with Potter's Wheel system, and [Lee+00], with IntelliClean tool.

In general, with respect to cleaning techniques, we found that we might classify them according to three dimensions: (i) approach, where we find techniques oriented to: rules, look-up tables comparisons, patterns, statistical methods, or functions (such as distance), (ii) user interaction degree, where we find tools that are interactive, automatic, a combination of both, or manual, and (iii) error type they manage, which were commented above.

In [Peralta-06], as commented in previous section, strategies for data freshness improvement are proposed. They analyze different solutions for the context of DIS that improve the freshness obtained in the data delivered to the user.

2.2.4 Summary of Data Quality Improvement

In this section we synthesize the existing knowledge about data quality improvement, basing on the previously analyzed works.

We find as general approaches to data quality maintenance, the following:

- Iteration of tasks for proactively improving quality: definition, measurement, analysis, improvement.
- Data quality monitoring and change detection.
- Models for the process applied to data, enhanced with data quality values representation, data quality checks, and corrections.
- Specific techniques for improving freshness and accuracy related quality factors in data.

With respect to analysis and improvement, we mainly find the following proposals:

- Analysis of causes of poor quality, mainly through the following approaches:
  - Detection of bottlenecks in the process that is applied to data
  - Comparison of data quality to benchmarks and comparison between different roles' perception of the data quality, as a possible source of errors.
  - Correlation between variables and analysis supported by domain experts
- Definition of patterns for improvement (problem - solution).

- Quality recovery through data cleaning and through modifications to the process (data-oriented or process-oriented techniques).
- Statement of information quality goals or user quality requirements.
- Improvement through notifications to the participant organizations in a cooperative system about better quality data that can be adopted by them.

With respect to data quality monitoring and change detection:

- Monitoring through on-line or off-line evaluations and change detection through comparison to quality requirements.
- In the context of cooperative information systems, offered data quality is monitored and when it changes it is notified to the participant organizations.
- Distributions of data values monitoring. Construction of baseline models and detection of quality changes through detection of deviations of data distributions from them.

With respect to models for data processing:

- Definition of models for the manufacturing process of an information product, which include modules for quality checkings and corrections.
- Definition of models that relate data quality, data quality requirements and processes that manage data.

With respect to specific techniques for improving quality in data:

- Data cleaning techniques, which identify a great gamma of possible errors in data, such as syntactic, semantic or consistency problems, and apply corrections basing on rules, look-up tables, patterns, etc.
- Strategies for improving freshness of data in the context of Data Integration Systems.

Finally, we classify some representative analyzed papers, according to some parameters that are relevant for our work, taking into account the most relevant characteristics of each proposal. Table 2.3 shows this classification.

Parameter	Option	Proposal
Context	Multiple data sources	[Cappiello+06-a] [Cappiello-05] [DaQuinCIS] [Peralta-06]
	Inside a corporation	[TDQM] [Scannapieco+02] [Caballero+04]
Quality Dimension	Only accuracy	[Madnick+01] [Galhardas+00] [Vassiliadis+01] [Raman+01] [Lee+00]
	Only freshness	[Peralta-06]
	In general, for various dimensions	[Cappiello+06-b] [TDQM]
General Improvement Approach	Iteration of tasks	[TDQM] [Cappiello+06-a]
	Monitoring and changes detection	[Cappiello+06-b] [Cappiello-05] [DaQuinCIS] [Bugajski+05-a]
	Construction of models	[Scannapieco+05-b] [Caballero+04]
	Specific improvement techniques	[Galhardas+00] [Vassiliadis+01] [Raman+01] [Lee+00] [Peralta-06]
Analysis of causes of poor quality		[Wang-98] [Lee+02] [Madnick+01] [Scannapieco+02] [Peralta-06] [Bugajski+05-a]
Approach for error detection and correction	Run-time	[Cappiello+06-b] [DaQuinCIS]
	Offline, through statistical models	[Bugajski+05-a]

Table 2.3: Classification of quality improvement proposals

3. Source Schema Evolution in DIS

We regard source schema evolution problem as a problem that may have some similarities with ours and that has been very much studied. Therefore we are interested in the general approaches that have been adopted in order to this problem. In the following we briefly comment some work about schema evolution and then we concentrate in the proposals for source schema evolution in DIS.

With respect to the general problem of schema evolution in databases we refer to some representative works. In [Zicari-91] and [Ferradina+96] two main aspects are taken into account in relation to the state of a database after schema evolution: (i) structural consistency and (ii) behavioural consistency. Structural consistency is the consistency between the database and the schema, and behavioural consistency is related to keeping the consistency of the application programs that existed before evolution. On the other hand, there are two approaches for managing schema evolution: (a) Adaptational approach [Ferradina+95] and (b) Versioning approach [Skarra+86][Ferradina+96][Lautemann-97][Nguyen+89]. In the adaptational approach, when the schema is modified the state of the schema before the change is lost and the final result of evolution

is an only one schema with the new structure. The existing instances and the application programs that run over the database have to be adapted to the new schema. In the versioning approach, modifications to the schema are not applied directly on the existing schema. Instead, a new version of the schema is created. In this case the existing instances do not necessary have to be transformed to satisfy the new schema, and neither the application programs. Finally, taxonomies for evolution and the effects of each operation on the schema and its instances are found in [Zicari-91] and [Skarra+86].

There are some works in the literature about source schema evolution in DIS, which we comment in the following.

A big amount of work has been done by Rundensteiner *et al*. in source schema evolution in the context of materialized views (MV). In [Rundensteiner+97] the authors present a study and classification of view adaptation problems. They distinguish the following problems: view synchronization, MV maintenance after view synchronization, MV maintenance after sources data updates, and MV maintenance after view redefinition. They also characterize the problem space in the view synchronization problem, considering a taxonomy of source schema changes, as well as the complexity of the view definition language and its meta-information. They propose a framework for solving view adaptation, called EVE (Evolvable View Environment), where there is a view synchronizer that rewrites the view definitions by replacing view components with suitable components from other ISs. In addition, in [Nica+99] they propose a language, called E-SQL, for evolvable view definition, which allows declaring if an attribute, relation or condition is replaceable and/or dispensable, and also characteristics about the new view extent with respect to the old one. The proposed strategy for view synchronization takes into account the evolution parameters imposed by the E-SQL view specification. Then they solve view maintenance after materialized view synchronization. Another algorithm for view synchronization is proposed in [Nica+98]. On the other hand, they have presented a work [Koeller+02] that proposes techniques for maintenance of schema-restructuring views. These views are defined with Schema-SQL language, presented in [Lakshmanan+96], which main characteristic is that allows a uniform manipulation of data and meta-data, generating a dynamic output schema.

The work presented in [McBrien+02] give solutions to source schema evolution in a DIS that is generated with a schema transformation approach. They propose a framework that supports evolution of source schemas allowing the global schema and the query pathways to be easily repaired. The same set of primitive transformations they propose for integrating source schemas into the global schema, are used for propagating source schema evolution.

Specifically in the context of Data Warehouse, we remark the works in [Bouzeghoub+00], [Bouzeghoub+03] and [Papastefanatos+07].

In [Bouzeghoub+00] the DW is defined, at the design level, as a hierarchy of view expressions whose ultimate nodes are queries on data sources. This schema is represented by a graph, and is used for design purposes as well as for managing evolution. The authors distinguish three different cases for DW evolution: (1) evolution of users' needs, (2) evolution of data sources and (3) evolution of the materialized views. For the case of data sources evolution they consider the changes: adding a new data source and deleting an existing data source. Both types of changes with the possible managements for them are analyzed in the paper. When it is possible, changes are solved at the graph without affecting the users' views. In the work presented in [Bouzeghoub+03] the system considered involves a mediation schema and a set of heterogeneous and autonomous data sources. The authors address the problem of propagating to the mediation queries changes raised at source schemas. They base themselves on the definitions of: a set of source change operations, a set of propagation primitives (modifications at the mediation level), and a set of

propagation rules. The rules determine which propagation primitives must be applied after a source change. They say that the propagation may either modify the mediation schema or the mediation queries, and they focus on the problem of propagation over the mediation queries. The proposed rules are ECA rules, whose events are stated in terms of source change operations, and whose actions are in terms of propagation primitives.

In [Papastefanatos+07] the authors propose an extension of the graph they proposed in previous works for representing ETL processes, such that they can represent the actions that should be taken when a change event occurs. They propose to enrich the ETL graph with annotations that facilitate what-if analysis. When an event occurs an action is triggered that either blocks the event or reshapes the graph to adapt to the proposed change. Three kinds of actions can be annotated for a construct of the graph (e.g. a query): propagate the change, block the change, prompt the administrator for deciding. They present an algorithm that determines how to propagate the change in the ETL graph.

Finally, relating DW evolution with quality, we find the work in [Quix-99], which provides a taxonomy of schema evolution operations and the quality properties that are affected by each of them.

As a general conclusion about the commented proposals for source schema evolution problem in DIS, we state that all of them mainly focus on the layer that maps sources and integrated system for absorbing the source changes. A wide gamma of strategies or approaches for data integration is managed in these works: materialized views, mediation queries, transformation primitives, ETL graphs. In all cases, the mapping generated between sources schemas and integrated schema is processed and managed according to the occurred change, in order to minimize the impact on the integrated schema.

4. Applications of Probabilistic Techniques to Data Management

In this section we briefly comment some works that, despite not addressing the same problem as we, they serve as a reference point for us with respect to the application of probabilistic models and/or techniques to data management problems.

In the following paragraphs we comment three works that address different problems, basing on probabilistic models. The first one, by Cho and Garcia-Molina, study how to estimate the change frequency of source data. The second one, by Karakasidis, Vassiliadis and Pitoura, models the refreshements of DWs using queue theory. The third one, by Liu, Luo, Cho and Chu, addresses the problem of selecting the most relevant database for a user query.

In [Cho+03] the authors consider a context were data sources are updated autonomously and the users do not know when and how often they change. The authors give some examples of applications that can improve their effectiveness using an estimated change frequency: a web crawler, the update policy of a DW, web caching, data mining. They study how to estimate how often a data item changes. A taxonomy is given, based on how the element is accessed (passive/active monitoring, regular/random interval) and what information is available (complete history of changes / last date of change / existence of change). They propose several estimators that measure the change frequency, assuming that a source element changes by a Poisson Process, in particular they mention experimental data that shows this behavior for web pages. They focus on estimating λ in the case that they only know whether the element changed or not between their accesses, and they present as further work the problem of changing λ.

In [Karakasidis+05], motivated by the need of as fresh data as possible in the DW, the authors propose a framework for the implementation of active data warehousing (DWs are updated as frequently as possible). In their architecture, data flows from the sources to the DW through an intermediate data processing stage, where it suffers different transformations. The authors employ queue theory as the cost model that predicts the data delay at this stage. They model each ETL activity as a queue in a queuing network, assuming that tuple arrivals to each ETL activity occur due to a Poisson process.

The work presented in [Liu+04] addresses the problem of metasearching, i.e. selecting the most relevant databases to a user's query on the Web. One of the techniques they propose is probabilistic relevancy modeling. They construct the probabilistic distribution for the relevancy of each database for a given query. Using the probabilistic model, the user can explicitly specify a desired level of certainty for the database selection.

There are some works that use probabilistic techniques for data cleaning. In [Chu+05] the authors, based on the fact that many applications exhibit strong dependencies between data samples, propose to use such dependencies for cleaning the data. Their approach is based on modeling data dependencies with Markov networks and use belief propagation to compute probabilities and to infer missing values or to correct errors. In [Andritsos+06] the authors address the problem of detecting duplicate tuples, corresponding to the same real-world entity. Their approach allows query answering over duplicated data, where each duplicate is associated with a probability of being in the clean database. For achieving this, they rewrite queries over the database containing duplicates.

In addition, there is a group of works that focus on the problem of uncertainty in databases, using in various different ways the potential of probability. For example, in [Cheng+05] probabilistic models are used for solving uncertainty of the database values, since they may not coincide exactly with the changing reality. It is also interesting for us, the inclusion to the queries of a probability requirement in the "where" condition. [Benjelloun+06], [Antova+07] and [Boulos+05] are other examples of probabilistic techniques application for managing uncertainty in databases' values.

5. How this Work Positions

We believe that our approach has important similarities with the approaches of Cappiello *et al.* and Bugajski *et al.*, which were presented in Section 2.2.2. The main points in common of our approach and Cappiello's is the objective of monitoring, and detecting quality changes in DIS, as well as the utilization of rules for their implementation. In particular, we coincide with one of their results that expresses that it is not necessary to undertake improvement actions every time the system identifies a quality problem (considering this has a cost). With respect to Bugajski's approaches we mainly coincide with the idea, strongly supported by them, of thinking on distributions, not on particular values. However, there are important differences between our work and theirs.

In the case of Cappiello's works, they detect data quality errors while we detect data quality changes. In addition, they focus on run time error detection and management, while we address offline changes detection and management. Finally, they work detecting errors in a one-by-one basis, while we model data quality behavior achieving the possibility of detecting and predicting changes on the behavior of data quality.

In the case of Bugajski's works, they probabilistically model the attributes' values, i.e. data values behavior, while in our work we probabilistically model the data quality values behavior.

The proposals presented in Section 4, specially the ones from Cho *et al.*, Karakasidis *et al.* and Liu *et al.*, reaffirm our approach of probabilistic modeling source behavior. At the same time, the proposal of Pon *et al.* (Section 2.1) is an example of a work that takes into account and emphasizes

the fact that data sources that have been accurate in the past are also likely to be accurate in the future. This concept is also used by us when working with quality models that are built from past behavior.

Evolution works were used as a starting point for analyzing the problem of quality changes, specially the problem of source schema evolution in DIS, which has some analogies with source quality changes in DIS.

It is very important to remark that the work presented in [Peralta-06] is a base from which we start for making our proposal. We use this work's results, in particular: quality factors study and definitions, and quality evaluation framework and algorithms, as a starting point. We believe that this work and ours are complementary.

6. Summary

In this chapter we presented an overview of the existing knowledge in the areas of data quality, source schema evolution in DIS, and applications of probabilistic techniques to data management.

Data Quality is a very wide research area. We commented the most important problems and some of the most relevant proposals for their solutions. For the problems of data quality dimensions and their measurement we have given a brief overview, while we concentrated on the issues of data quality evaluation in DIS and data quality improvement in general.

For data quality evaluation we highlighted the strategies for combining quality values that come from different sources. From the proposals for data quality evaluation, we extracted the concrete techniques for accuracy and freshness evaluation, which are the aspects we must take into account in our work. We found and selected some proposals for accuracy and only one for freshness.

Data quality improvement was the area we analyzed more in depth, since it is the closest one to our work. Works that address quality maintenance and quality changes management are usually presented as quality improvement works. Besides, data quality maintenance includes the problem of data quality improvement.

The approaches of the analyzed improvement works basically focus on any of the following: (a) iteration of tasks for proactively improving quality, (b) data quality monitoring and change detection, (c) models for the process applied to data, enhanced with data quality issues, (d) specific techniques for improving quality factors in data. We classified the works into three categories: practices-oriented improvement, data-processing-oriented improvement, and specific quality improvement techniques. The works classified in the second category are the ones that are closest to ours, and their approaches have points in common with ours. In Section 5 we position our work with respect to these ones.

Some important proposals about source schema evolution in DIS were commented in order to show some basic knowledge about how to manage this problem.

Finally, works about probabilistic techniques applications to the area of information systems are commented. Some of these works show how data behavior is probabilistically modeled, and in particular, there are cases that are assumed to behave as a Poisson process.

As we tried to remark in this chapter, many approaches throughout the analyzed literature present characteristics that show the pertinence and relevance of the approach we have chosen in the present work.

CHAPTER 3. MAINTAINING QUALITY

We should take care of quality in our Data Integration System, avoiding its degradation.

1. Introduction

As said before, our goal is to maintain the system quality at a level that satisfies the user quality requirements. In order to achieve this, we must manage all the changes that affect system quality, considering that it may be continuously changing, in particular its data sources, its data transformation graph and its users' requirements.

For the management of DIS changes we identify two possible approaches. In the first one, we simply act when a change occurs, evaluating if it provokes the dissatisfaction of the user quality requirements and acting in consequence. We work only with what we see in a certain moment (a "snapshot" of the system). This strategy would manage each quality change independently, using the available information about the current state of the system. It allows solving the quality problems originated by each change on the system. However, when thinking on system changes, it is clearly more natural and useful to have a dynamic vision of the system. This is the base of the second approach. It has a vision of the system through time, and not only in a punctual moment. This dynamic vision allows to better diagnose what is happening to the system and to take more effective decisions. The main advantages of having this vision are the possibilities of: (i) detecting more "macro" changes, for example a change on the average freshness of a source, instead of a change on the source freshness in certain moment, (ii) acting preventively, i.e. taking actions before the system is negatively affected by a change, and (iii) considering historical information for determining the actions to be taken after a change. The second one is the approach we choose for maintaining quality. We present the following simple examples with illustration purposes.

Example 3.1 (about freshness)

A DIS has a data source that provides it economic information, whose freshness oscillates between 0 and 12 hours. However, only in exceptional cases, when the enterprise does not work normally due to an unexpected holiday or employees' problem, the data source freshness reaches 13 hours or more. The DIS does not satisfy users' freshness requirement if this source passes the value of 12 hours. Suppose that the first approach for changes management is being applied. One day, the source reaches 13 hours, this change is detected by the DIS quality management system, and as a consequence the source is eliminated from the system, and substituted by another one. In fact, if the DIS administrator would have known that this was an exceptional situation, he would not have taken that decision since the source was very reliable and accurate.

◊

Example 3.2 (about accuracy)

Suppose there is an operational database that registers the sales in certain store, and which also serves as a data source for a DIS of the global company. There are pre-defined procedures for measuring the accuracy of the DIS-sources' data, which are executed periodically at the sources. In certain moment this measurement is applied to the mentioned store database and the resultant accuracy is under the acceptable values for the accuracy required at the DIS. The DIS administrator immediately starts implementing very expensive

cleaning processes for the data extracted from this source (affecting the obtained freshness). Perhaps, if the DIS administrator would have known that the last-measured accuracy was strongly influenced by one exceptional day, when the operator was momentarily substituted by an inexperienced one, he would not have taken that decision.

◊

With these examples we intend to show the importance of determining which situations should be considered as relevant quality changes, since considering a change as relevant may have drastic consequences on the DIS, its design, its sources, etc. One of our main sub-goals is to propose good criteria for determining which changes are relevant and an effective mechanism for identifying them. Once a relevant change is detected, it must be determined which are the possible actions that may be taken in order to "repair" the DIS quality. For this, having a dynamic vision of the DIS quality is also very useful, since it gives more information for selecting the best actions to be taken.

We carry out the chosen approach strongly basing on probabilistic and statistical techniques.

Quality maintenance is performed by the Quality Management System, which is in charge of all the functionalities related to the quality of the DIS. It is in charge of quality evaluation, proposed in [Peralta-06], quality-oriented DIS design, partially addressed in [Peralta-06] and in the present work, and quality maintenance, addressed in the present work.

In this work we focus on the management of two factors, freshness and accuracy, considering definitions and evaluation-calculations that are simple or simplified. This is because our goal is to propose techniques for changes management and we prefer as a strategy, to start with non complex factors and calculations such that the focus can be posed on the object of our study.

In this chapter we intend to give the support for understanding the proposal of the thesis, which is presented in the three following chapters. We present: a set of concepts that are basic in our work, the specification of the basic components of our framework, and the whole mechanism that we propose for maintaining quality. The latter is necessary to put the pieces together, because we present the solutions to the different aspects of the problem in different chapters. These are: the modeling of quality behavior, in Chapter 4, the detection of relevant quality changes, in Chapter 5, and the repair of DIS quality, in Chapter 6.

In Section 2 we present basic definitions, in Section 3 we present the mechanism for maintaining quality and in Section 4 we present the summary of the chapter.

2. Basic Definitions

In this section we present some concepts that are basic in our work. Some of them are taken from existing work, some are adapted to our problem, some are proposed by us. We also present here the specification of some of these concepts, which will be extended all through the document.

2.1 Data Integration System and Quality Management System

As said before, the *Data Integration System* (DIS) is an information system that integrates data from a set of heterogeneous and autonomous information sources and provides it to users. In our context, it basically consists of a set of data sources, a transformation process that is applied to data extracted from sources, and a user front-end, which is a set of pre-defined queries or an integrated schema. The data involved in these different elements have some quality, which can be measured, estimated, and eventually improved.

We propose the existence of a *Quality Management System* (QMS) associated to the DIS. The main functionalities of this system are the following:

- Construction of source quality model.
  Given certain information about the source, the QMS is capable of building a model that represents the quality behavior of the source.
- Evaluation of the quality of the DIS. (Proposed in [Peralta06])
  The quality values provided by the DIS to the users are calculated from the sources quality values.
- Construction of the DIS quality model.
  Given the sources quality models and the user quality requirements, the DIS quality model is calculated.
- DIS quality changes management.
  The QMS is capable to detect relevant quality changes and to propose actions in order to repair the DIS quality.

The QMS functionalities are based on a framework for quality management. This framework, called *Quality Management Framework*, is an extension of the *Quality Evaluation Framework* proposed in [Peralta-06].

2.2 Freshness and Accuracy

As seen in Chapter 2, there are many different interpretations and definitions of freshness and accuracy quality factors. We intend to give solutions that are as general as possible for freshness and accuracy factors, however, in order to avoid any ambiguity, we choose the following particular definitions for these factors and we restrict ourselves to them in all our proposals.

Definition 3.1: *Freshness* is the time elapsed since the data is updated at the DIS sources until the data arrives to the DIS data target (user-query answer or materialized database).

Definition 3.2: *Accuracy* tells how correct the data is. It may refer to any of the following interpretations: *semantic correctness*, *syntactic correctness* or *precision*, according to their definitions in [Peralta-06]. Basically, semantic correctness describes how well data represent states of the real-world, syntactic correctness expresses the degree to which data is free of syntactic errors (such as misspellings and format discordances), and precision concerns the level of detail of data representation.

For our work we assume some context characteristics that are related to the management of these quality factors in the DIS, which are the following:

- Granularity. The granularity is the basic information unit to which quality measures are associated. We manage a granularity of relation. At the sources we have a quality value for each source relation (in general, we call it source), in the transformation process we have a quality value for each activity result, which is a relation, and we also have a quality value for each data target, which is also a relation.
- Measure unit.
  - In the case of freshness, it is a Natural number that represents any measure of time; days, hours, minutes, seconds, etc., according to the real case.
  - In the case of accuracy, it is a Decimal number between 0 and 1, whose precision is determined according to the needs of the real case.

- Time and accuracy values are discretized as explained later, in Chapter 4, Section 2.2.

o Measurement at the sources.

- In the case of freshness, it is measured in function of the occurred updates. When there is an update in a source relation it holds freshness = 0, as time passes, freshness increases one by one according to the chosen unit.
- In the case of accuracy, there are processes that measure the different types of accuracy in a relation. In general, it is measured cell[1] by cell and these measures are aggregated to a granularity of relation [Etcheverry+06].

o Information about sources. The Quality Management System needs metadata from the sources that provide either the quality values or data that allows deducing them.

- In the case of freshness, the system receives information of the updates occurred on the source or information that allows estimating the freshness through the construction of a model.
- In the case of accuracy, either the source provides the accuracy values of each relation or the system itself measures them periodically.

o User required values. The user expresses the freshness and accuracy values with the same units, precision and interpretation we presented above.

2.3 Quality Evaluation

Quality evaluation problem in DIS is addressed in [Peralta-06]. We incorporate the proposed solutions and, as said before, extend the proposed framework (Quality Evaluation Framework).

The proposed techniques for quality evaluation are applicable in different situations and with different strategies. In the following sub-section we state our approach for their application.

2.3.1 DIS Quality Evaluation through Estimations

We identify two different ways of evaluating quality in a DIS. We call them calculation and estimation. In the case of calculation, at the moment of integrated schema population or at the moment of user-query execution, quality is measured at the sources and propagated (combined and calculated) with the data that goes from the sources to the user. In the case of estimation, sources quality is measured and DIS quality (quality of information provided by the DIS) is estimated, regularly. When the user comes to use the DIS, it provides quality information before he retrieves the data from the sources. This quality information is an estimation based on knowledge about sources quality (which may be obtained in different ways) and about sources and DIS characteristics. Estimated quality is the quality that the DIS is currently providing (at any moment DIS information is required), while calculated quality is the quality provided by the DIS at certain instant with a certain dataset. Calculation needs information that is only obtained through execution of the data queries or transformations. Table 3.1 shows a summary of Calculation/Estimation characteristics.

In this work we consider that the DIS quality is evaluated through estimation. In this context, we do not give the quality values of certain data, we give an estimation of the data quality that the DIS is offering to its users. That is why we manage average, maximum, most probable, etc. quality values,

[1] We call cell to the value of an attribute in a tuple of a relation.

instead of punctual quality values. If we considered, for quality evaluation, calculation instead of estimation, it would not have any sense to talk about changes, since quality would be fixed values corresponding to certain data at a certain moment. We would not have any changes to detect or calculate their impact.

Calculation	Estimation
Based on actual quality values of the sources	Based on representative quality values of the sources (max, average, etc.)
Using actual sources' instances	Using usual characteristics of sources' instances
Actual quality values of the DIS are calculated	Mean, maximum or most-probable quality values of the DIS are calculated
Quality given by the DIS at a given time instant	Quality given by the DIS at any moment

Table 3.1: Calculation vs. Estimation

2.3.2 Quality Evaluation Framework

The Quality Evaluation Framework is proposed in [Peralta-06], first as a specific solution for freshness quality factor and then it is extended for accuracy factor. In the framework the DIS is modeled; sources, transformation process and data provided to users are modeled.

DIS is modeled as a workflow process in which the workflow activities perform the different tasks that extract, integrate and convey data to end-users. Quality evaluation algorithms are based on the workflow's graph representation, basically consisting on value aggregation and propagation through this graph.

The following definitions are textually transcribed from [Peralta-06], with the exception of Definition 3.3, where we substitute the set of QualityGraphs by only one QualityGraph. This is done with the purpose of simplifying the specification and does not interfere with the correct DIS representation.

Definition 3.3: The *quality evaluation framework* is a 5-uple: <*Sources*, *Targets*, *QualityGraph*, *Properties*, *Algorithms*>, where *Sources* is a set of available data sources, *Targets* is a set of data targets, *QualityGraph* is a graph representing the DIS process, *Properties* is a set of properties describing DISs features and quality measures and *Algorithms* is a set of quality evaluation algorithms. □

Definition 3.4: A *data source* is represented by a pair <*Name*, *Description*>, where *Name* is a String that uniquely identifies the source and *Description* is a free-form text providing additional information useful for end-users to identify the source (e.g. URL, provider, high-level content description). □

Definition 3.5: A *data target* is represented by a pair <Name, Description>, where Name is a String that uniquely identifies the data target and Description is a free-form text providing additional information useful for end-users to identify the target (e.g. application/process name, interfaces, servers running the application). □

It is introduced the concept of *quality graph*, which is a graph that has the same workflow structure as the DIS and is adorned with additional DIS information that is useful for quality evaluation. Figure 3.1 shows an example of quality graph.

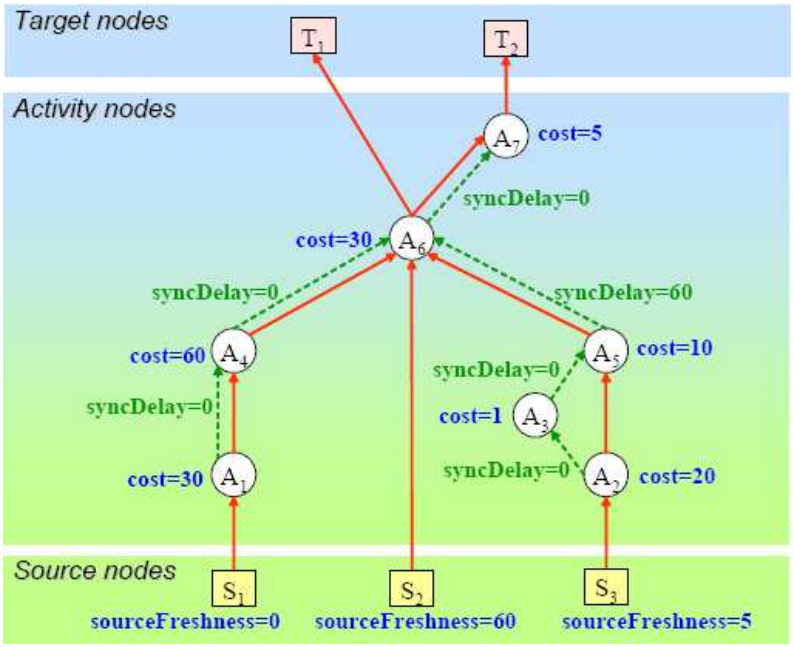


Figure 3.1: Quality graph [Peralta-06]

2.4 Specific Scenarios for Managing Freshness

Freshness quality factor is highly dependant on certain context characteristics that generate different scenarios. We define three dimensions that are relevant to freshness management and whose crossing generates the different scenarios. The dimensions are the following:

1) DIS materialization

   This dimension refers to the degree of materialization of the integrated schema. This is very important when considering freshness property, since it directly affects the calculation of the freshness of the data that arrives to the user. The DIS are classified in (i) Virtual, (ii) Materialized, and (iii) Hybrid, as presented in Chapter 1, Section 1.

2) Sources loading

   In order to know about the behavior of freshness factor at a source, it is essential to know how the source is loaded. In this dimension we classify the sources according the way they are loaded, into two categories: (i) periodic loading, and (ii) continuous loading.

   In (i) the source is loaded or updated in a periodic basis, i.e. the data is always updated with exactly the same frequency. An example of this kind of source may be a Data Warehouse that is loaded every month, and provides its information to a system that integrates data from several Data Warehouses.

   In (ii) the source is updated in a continuous basis, randomly. This occurs, for example, with sources that are autonomous with respect to the integrated system, and are the operational databases of other systems.

3) Sources meta-data

We distinguish four kinds of meta-data, related to source freshness, that may be provided by the sources: (i) date-time of last update, (ii) all or some of the occurred updates, (iii) update period (only for case of periodic loading), and (iv) estimations of the update frequency.

Example 3.3

We classify, according to the previous dimensions, a Data Warehouse and an OLAP System.

The Data Warehouse has a materialized integrated database, its sources are updated in a continuous basis, since they are operational databases, and each source provide to the Data Warehouse system the estimation of the frequency of updates.

The OLAP system integrates information coming from several Data Marts, which are its sources. It has a virtual integrated database, its sources are updated periodically, and they provide information about the update period. (In [Marotta+06] we propose techniques for the management of quality factors in a Data Marts system with these characteristics.)

Figure 3.2 shows this classification by the different dimensions.

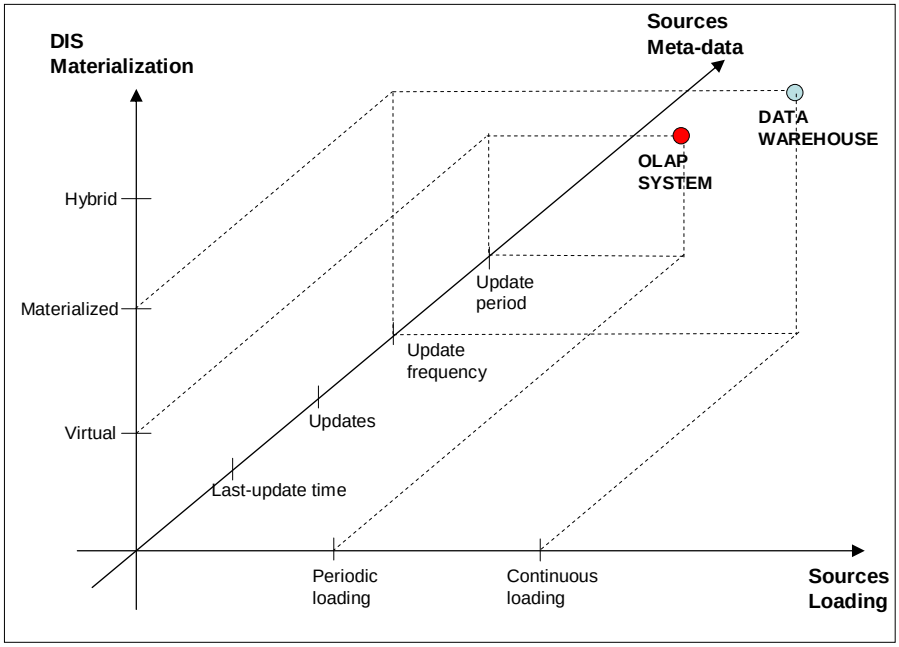


Figure 3.2: Classification of example cases

◊

3. Mechanism for Maintaining Quality

In order to maintain the quality of the DIS at a level that satisfies user quality requirements, the QMS must be **continuously** doing two basic tasks: (1) maintaining up-to-date meta-information about system quality and about other system properties that affect quality, and (2) monitoring the quality of the system (this includes the quality of the sources and the quality of the DIS), detecting when there is a change that is relevant to DIS quality. On the other hand, the QMS must **occasionally** react doing the following tasks: (3) evaluations and calculations related to DIS current quality, as necessary and (4) analysis and recommendations of actions for repairing DIS quality.

3.1 Mechanism overview

The mechanism for maintaining quality proposed in the present work is integrated with other functionalities into a larger context. This context is composed by the Data Integration System (DIS), meta-information about it, the Quality Management System (QMS), and the users. Figure 3.3 shows the architecture of the whole process of DIS quality management, which we explain in the following paragraphs.

As can be seen, in the DIS environment we distinguish the DIS processes, and other four processes that interact with it: *Execution*, which obtains information about the DIS processes execution, *Event Monitor*, which monitors the events sent by the sources, *Design & Maintenance*, which executes design tasks and modifications to the DIS, and finally, *Measurement*, which is a process that may exist for measuring the quality of data extracted from a source.

The meta-information needed by the QMS acts as an interface between the DIS environment and the QMS. We divide it in the *events repository* and the *estimations & statistics*. The first one stores all the events that it receives, which come from the Event Monitor and/or triggered by changes on *estimations & statistics*. The second one stores a large variety of information about the system, which we can group basically in: current information and estimations about DIS properties, quality models, historical information about them, and statistical data about quality measurements.

In the QMS we distinguish three groups of processes, one related to the building of the management framework, a second one related to the evaluation of DIS quality, and a third one related to the improvement of DIS quality. All the processes of these groups interact with the Quality Graph, which is the representation of the DIS.

Quality Graph Builder is in charge of the creation and maintenance of the quality graph, obtaining the necessary information from the *estimations & statistics* metadata. *Quality Models Builder* calculates the quality behavior models of the sources and the DIS, obtaining information from the meta-information repositories and storing the models and the histories of these models in *estimations & statistics* repository.

Quality Changes Detection and *Quality Verifications* are in charge of managing the changes occurred on the DIS, and determining if they are relevant to the DIS quality. *Quality Verifications* verifies the DIS quality taking into account the user quality requirements and the quality provided by the DIS to the user. *Quality Evaluation* evaluates the quality given by the DIS, providing this information to the end-user if needed.

Design Refinement improves the DIS design basing on quality evaluation and an analysis of critical points. *Quality Repair* is the process that compensates the DIS quality when it was negatively affected by a change, and this event was notified to it by the *Quality Changes Detection* process. The *Improvement* processes do not act directly on the DIS, but they give the designer the recommendations of which improvement actions he should apply.

Not all the solutions for the different modules shown in the figure are given in the present work. Here we propose solutions for the processes: *Quality Models Builder*, *Quality Changes Detection*, *Quality Verifications* and *Quality Repair* (green-colored in the figure), we define meta-information about *events* and *estimations & statistics*, and we enrich the *quality graph* specification. The processes *Quality Graph Builder*, *Quality Evaluation* and *Design Refinement*, as well as the *quality graph* proposal and specification, are addressed in [Peralta-06].

Our proposal for the process *Quality Models Builder* and the meta-information it manages is presented in Chapter 4. Our proposal for *Quality Changes Detection* and *Quality Verifications*, as well as the events they manage, are presented in Chapter 5. Our proposal for *Quality Repair* is

presented in Chapter 6. Our specification of the management framework is presented in next section of this chapter.

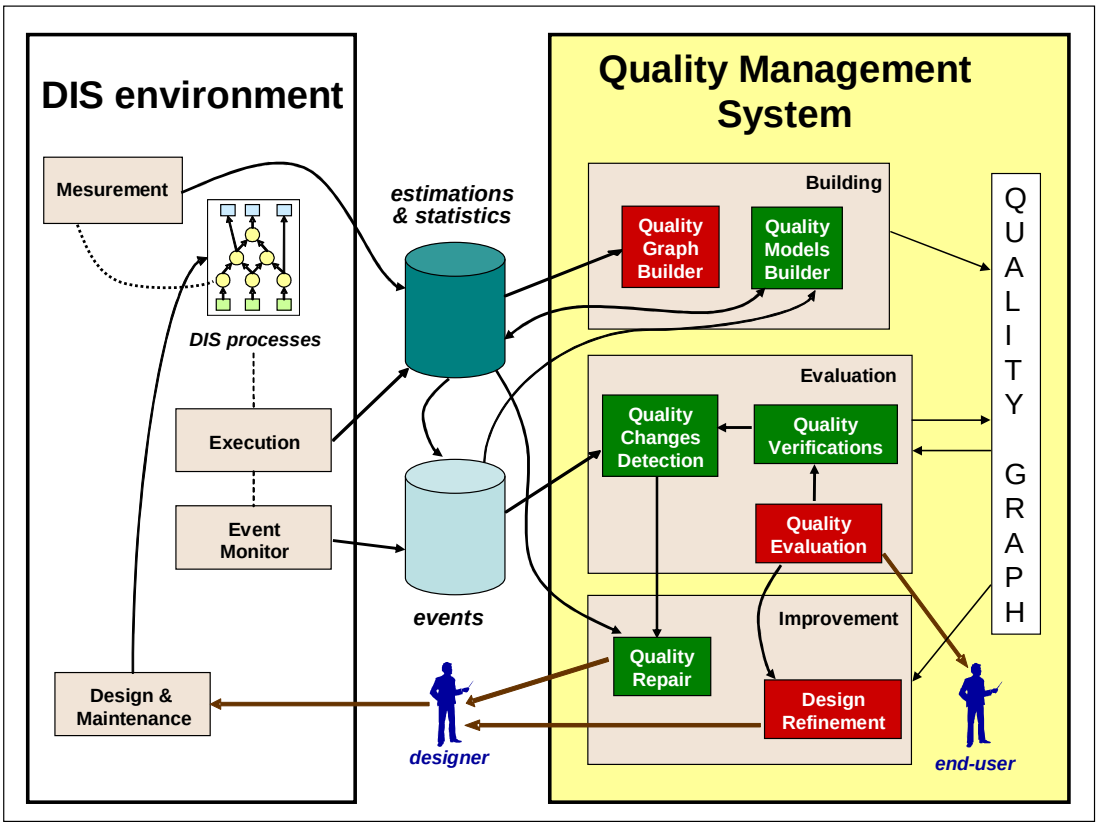


Figure 3.3: Architecture of the process of DIS Quality Management

The parts of the architecture covered in our work are shown in Figure 3.4. There we also show how the modules of DIS quality maintenance are inter-connected. *Quality Models Builder* feeds *estimation & statistics* repository, which generates *events* (through database triggers), which are received by *Quality Changes Detection*, which in some cases invokes *Quality Repair*.

The mechanism we propose for maintaining quality is supported by two main actions: (1) up-to-date maintenance of the quality models and statistics, and (2) management of the DIS changes that affect quality.

Quality models of the sources are maintained and monitored such that changes on them can be detected. This maintenance is done by *Quality Models Builder* based on events that come from sources, and on periodic measurements. The monitoring is automatically carried out by *estimation & statistics* database, through triggers that react when models are updated.

In addition to source quality models changes, changes on user quality requirements and DIS transformation process are detected. These changes are notified to *Quality Changes Detection* module through events, which are generated from *estimation & statistics* repository.

Quality Models Builder is also in charge of maintaining DIS quality models. These models take into account sources models, DIS transformation process, and user quality requirements. They are used for determining if the DIS quality is acceptable or if it must be repaired.

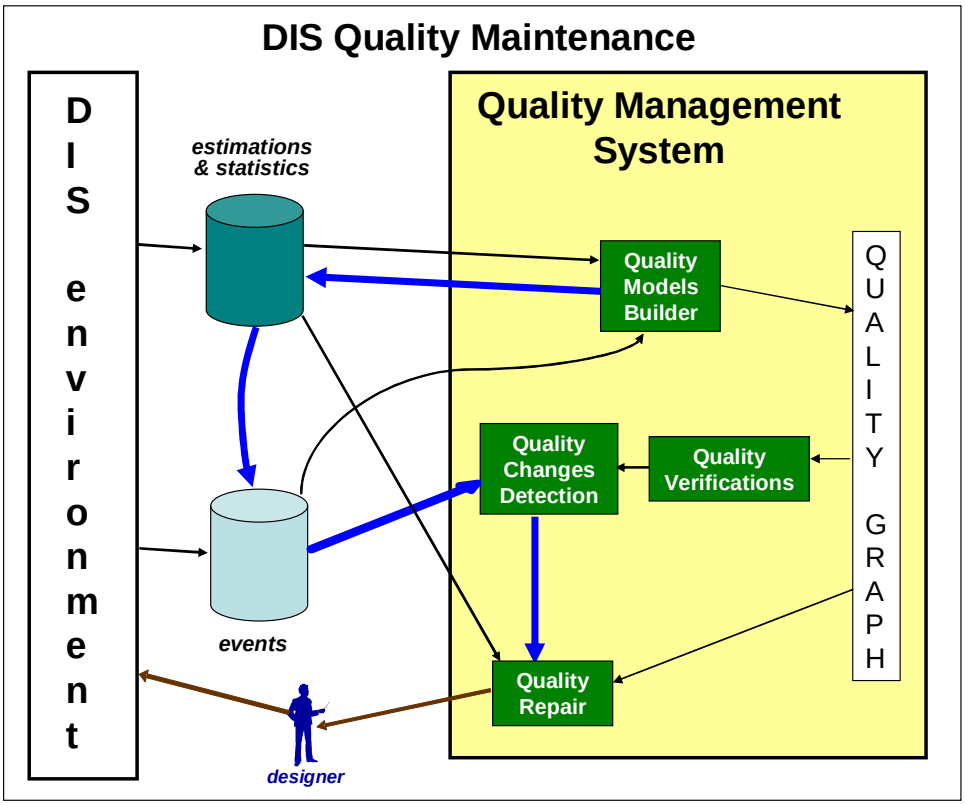


Figure 3.4: Modules of DIS Quality Management solved in the present work

It is important to note that in the proposed mechanism actions are taken in function of changes that have already occurred. If a quality factor reaches a value that falls far from the expected ranges, this situation is not avoided neither managed at the instant it occurs. The reached value affects the general quality behavior of the sources and system, and it is later taken into account in the relevant changes detection.

3.2 Quality Management Framework

In order to specifying and implementing our solutions we define a framework, called Quality Management Framework. In the following we present its specification. We extend the Quality Evaluation Framework of [Peralta-06], also making some modifications to the existing specifications. Our definitions are based on the existing definitions, changing basically the way the *properties* are defined and related to the quality graph or its elements. This modification is necessary because in our proposal most of the managed concepts are represented as *properties* and the previously existing definitions do not adapt to our needs naturally.

Definition 3.6: The *Quality Management Framework* is a 6-uple QMF = <Sources, Targets, QualityGraph, Properties, Algorithms, ChangeManagementElements>, where.

- Sources is a set of available data sources,
- Targets is a set of data targets,
- QualityGraph is a graph representing the DIS process,
- Properties is a set of functions for describing DISs features and quality measures,

- Algorithms is a set of quality management algorithms,
- ChangeManagementElements are the tools that are used to manage quality changes. □

Definition 3.7: The *ChangeManagementElements* is a 7-uple QMF = <Events, DetectionRules, Statistics, Interpretations, InterpretationRules, Actions, RepairingRules>, where.

- Events is a set of events managed by the QMS,
- DetectionRules is a set of rules that allow detecting relevant quality changes,
- Statistics is a set of functions that give current and historical information about the DIS,
- Interpretations is a set of interpretations about DIS situation,
- InterpretationRules is a set of rules that deduce interpretations from events and statistics,
- Actions is a set of actions that may be applied to the DIS,
- RepairingRules is a set of rules that define, from the interpretations, which actions should be taken for recovering DIS quality □

The different components of *ChangeManagementElements* are defined and specified through the different chapters of this document, except in the case of *Statistics* component, which is not completely specified in this work. It involves many different kinds of meta-information, which in some cases is specified and in other ones not. In all cases this information is assumed to be available and up-to-date in the *estimations & statistics* repository.

Definition 3.8: A *data source* is represented by a pair <*Name*, *Description*>, where *Name* is a String that uniquely identifies the source and *Description* is a free-form text providing additional information useful for end-users to identify the source (e.g. URL, provider, high-level content description). □

Definition 3.9: A *data target* is represented by a pair <Name, Description>, where Name is a String that uniquely identifies the data target and Description is a free-form text providing additional information useful for end-users to identify the target (e.g. application/process name, interfaces, servers running the application). □

Definition 3.10: *PropTypes* is a set of Strings, each of which names a property type. A property type is for example, “user quality requirement”, “cost”, “source quality behavior model”. □

Definition 3.11: *PropertiesDefinition* is a function *PD*: PropTypes → U that gives for each property type the domain of the properties of the type, where U is the universe of domains (the set of possible domains for a property). □

The quality graph specification we propose has two main differences with the one of [Peralta-06]: we specify a relation between graph elements and property values and we add the possibility of assigning properties globally to the graph.

Definition 3.12: A *quality graph* is a 5-uple G = (V, E, ρ_V, ρ_E, *gp*) where:

- V is the set of nodes. V^s, V^t and V^a are the sets of source, target and activity nodes respectively; with $V = V^s \cup V^t \cup V^a$. Each source or target node corresponds to a source or target of the framework.
- $E \subset V \times V \times T$ is the set of edges. T ={c, d} distinguishes between control edges (c) and data edges (d). The edge $(u, v)^t$ originates at node *u*, terminates at node *v* and has type *t*; with $u, v \in V$, $t \in T$.
- ρ_V represents the node properties. It is a function that given a node returns a function, which given a property type returns a function, which given a property name returns the value of the property,

  $\rho_V : V \rightarrow \{f / f: \text{PropTypes} \rightarrow (\text{String} \rightarrow \mathcal{U}) \wedge$

  $\forall x \in \text{PropTypes}\ \forall y \in \text{String}\ (f(x))(y) \in PD(x) \cup \{\perp\}\},$

  where $\mathcal{U}$ is the union of the property domains.
- ρ_E represents the edge properties. It is a function that given an edge returns a function, which given a property type returns a function, which given a property name returns the value of the property,

  $\rho_E : E \rightarrow \{f / f: \text{PropTypes} \rightarrow (\text{String} \rightarrow \mathcal{U}) \wedge$

  $\forall x \in \text{PropTypes}\ \forall y \in \text{String}\ (f(x))(y) \in PD(x) \cup \{\perp\}\},$

  where $\mathcal{U}$ is the union of the property domains.
- *gp* represents the graph properties. It is a function that for each property type, gives a function that for each property name gives the value of the property,

  gp: PropTypes $\rightarrow \{f / f: \text{String} \rightarrow \mathcal{U} \wedge$

  $\forall x \in \text{PropTypes}, \forall y \in \text{String}, (gp(x))(y) \in PD(x) \cup \{\perp\}\},$

  where $\mathcal{U}$ is the union of the property domains. □

It is considered, without loss of generality, that target nodes have a unique incoming data edge and source nodes have a unique outgoing data edge.

To better understand the representation of properties in the quality graph, refer to Example 3.4, shown in the following subsection.

3.3 Quality Requirements

The use of statistical and probabilistic values for DIS quality factors gives the possibility of expressing the user quality requirements using also these kinds of values.

We define the following types of user quality requirements that can be expressed in the DIS:

- value + probability

  The user gives a quality value v and a probability value p. This means that the user requires a maximum or minimum (depending on the quality factor) value v, which must be verified by the DIS with a probability p. In other words, quality value v must be verified in the p*100 percent of the cases.

- maximum / minimum

The user expresses that the quality value of the DIS element must be greater than or equal to or less than or equal to a given value.

- average

  The user expresses that the average quality value he expects from the DIS element, must be the given value, ≤ or ≥, according to the quality factor.

- most frequent value

  The user expresses that the most frequently quality value verified by the DIS element, must be the given value, ≤ or ≥, according to the quality factor.

These possibilities for expressing quality requirements give to the user much more flexibility than expressing the requirements only by a quality value. For example, suppose a user needs that certain information has a freshness of, as maximum, 24 hours, but he does not care if in some punctual cases this freshness reaches greater values. He can state as quality requirement: freshness <= 24 with probability = 0.9. With this requirement he accepts greater freshness values in 10% of the cases, otherwise, if he could not express this, perhaps he would loose some source, which cannot verify always the requirement, or he would accept a maximum value of 48 hours.

Definition 3.13: *User Quality Requirement* is a property type, whose corresponding domain is a set of 4-uples of the form: <qfactor, type, value, prob>, where:

- qfactor is a String, representing the quality factor.
- type ∈ {"probability", "maximum", "minimum", "average", "frequency"), tells the type of the requirement.
- value is a Decimal, the quality value of the requirement.
- prob ∈ 0..1, is the probabilistic value associated to the quality value. Used only when type = "probability". □

We present an example in order to illustrate the representation of these properties in the DIS.

Example 3.4

Suppose the data target T1 has four user quality requirements defined, two for freshness and two for accuracy. They are specified as follows:

$((\rho_V (T1))$ (UserQualityRequirement)) (req1) = <freshness, maximum, 10, NULL>,

$((\rho_V (T1))$ (UserQualityRequirement)) (req2) = <freshness, probability, 8, 0.9>,

$((\rho_V (T1))$ (UserQualityRequirement)) (req3) = <accuracy, minimum, 0.7, NULL>,

$((\rho_V (T1))$ (UserQualityRequirement)) (req4) = <accuracy, frequency, 0.9, NULL>

For referencing all T1 requirements:

T1Reqs = {<n, r> / n ∈ String, r ∈ *PD*(UserQualityRequirement) and

$r = (\rho_V (T1)$ (UserQualityRequirement)) (n)}

◊

It is well worth clarifying that the requirement types "average" and "frequency" are later confronted with the calculations of *expectation* and *mode* (concepts that are explained in next chapter),

respectively, of the sources quality models. When referring to user requirements we prefer to use the terms “average” and “frequency” because they are closer to the user perspective.

4. Summary

In this chapter we present different concepts that are basic for the comprehension of the following chapters. They are basic concepts, definitions, specifications of objects that are managed in successive chapters, and a complete vision of the mechanism proposed in this thesis.

The basic concepts we present are those definitions and assumptions from where we start. Some are taken from previous work, some are taken and adapted to our needs, and others are defined by us.

The mechanism for maintaining quality that we present can be seen as the summary of our proposal. It contains the different solutions we give in this work and how they inter-relate. In addition, we present the specification of the framework we use in the rest of the work and the definition and specification of the user quality requirements.

CHAPTER 4. QUALITY BEHAVIOR MODELS

How does quality vary in our Data Integration System?

1. Introduction

In order to achieve the dynamic vision of the system quality and considering the volatility of the source quality values, we build quality behavior models for the sources as well as for the system itself. The quality behavior models are strongly based on probability distributions. Having probabilistic models describing the quality of the sources and the DIS gives the possibility of foreseeing what may happen with quality in the short term.

The idea is to model which can be the quality-factors' values at any instant and how they vary through time. In the case of the sources, for each source we calculate the probability distribution of the quality values. In the case of the system, we calculate the probability that its quality satisfy the required quality, and the probability distribution of the quality values given by the system. In both cases we maintain the history of the models.

In order to buid and maintain these models we monitor the sources and the DIS and we maintain metadata, which constitute the support for the DIS quality maintenance.

In Section 2 we present an overview of the probabilistic concepts we apply, en Section 3 we present the modeling of the sources quality, in Section 4 we present the modeling of the DIS quality, in Section 5 we comment the strategy for having the information about quality through time, and finally, in Section 6, we present the summary of this chapter.

2. Probabilistic Techniques Application

As said before, we want to model the system quality as something dynamic, and we achieve this through the construction of probabilistic models.

Given some information relative to the behavior of the quality factor, the probabilistic techniques application allows us to: have an estimation of the current quality value and to have information about the possible quality values in the short term. These techniques give enough information to characterize the quality of a source as well as the quality of the DIS.

In this section we enumerate and briefly explain some probabilistic concepts that we apply in our problem, and we comment some particularities of this application.

2.1 Some Probabilistic Concepts

We give here an overview of the basic concepts we apply in our analysis and solutions, mainly based on [Canavos-88][IOCourse-07][Virtamo-04][Wikipedia-07]. The idea is to briefly comment these concepts, assuming the reader has a basic knowledge of the topic. For a deeper study, the interested reader may go to the cited references.

Random Experiment

It is a process whose result can be observed, but whose value cannot be anticipated with certainty. Each possible experiment result is called random point and noted w_i.

Sample Space

It is the set of all the possible results of a random experiment.
$\Omega = \{w_i: w_i \text{ is an observable result of the experiment}\}$

Event

It is any sub-set of the sample space.
An event *A* consists in the set of the random points w_i that represent experiment results where *A* occurs.

Union of E_1 and E_2: $E_1 \cup E_2$. The event that consists of all the possible results of E_1, E_2 or both.

Intersection of E_1 and E_2: $E_1 \cap E_2$. The event that consists of all the results that belong to E_1 and E_2.

E_1 and E_2 are ***disjoint*** if $E_1 \cap E_2 = \varnothing$

Probability

The probability is a real number that measures the possibility that a result of the sample space occur when the experiment is carried out.

There are three different interpretations of the probability: *classic*, *of relative frequency*, and *subjective*. The first and second ones are based on the repetition of experiments carried out under the same conditions. The third one represents a measure of the degree of belief with respect to a proposition.

The axiomatic definition of Probability is the following:

> Let S be a sample space and E any event of S. P(E) is a probability function over S if it satisfies the following axioms:
>
> 1) $P(E) \geq 0$
> 2) $P(S) = 1$
> 3) If, for events $E_1, E_2, E_3, \ldots$
>
> $E_i \cap E_j = \varnothing$ for all $i \neq j$, then
>
> $P(E_1 \cup E_2 \cup \ldots) = P(E_1) + P(E_2) + \ldots$

The ***addition rule of probabilities*** is used for any two events E_1 and E_2 of the sample space:

> $P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$
>
> Intuitively, $P(E_1)$ and $P(E_2)$ reflect the number of times that results of E_1 and E_2 will occur, respectively. It is necessary to subtract the common results because otherwise they are double counted.

Joint probability is the probability of the intersection of two events. It is the probability of two events in conjunction.

Marginal probability is the unconditional probability of an event, the probability of one event regardless of the other event.

Conditional probability is the probability of an event, conditioned to the previous occurrence of other event.

> Let A and B be two events of the sample space S, so that P(B) >0. The conditional probability of A when B occurs is the following:
>
> $P(A \mid B) = P(A \cap B) / P(B)$

Statistical Independence

Let A and B be any two events of a sample space. A is statistically independent from B if P(A|B) = P(A). This has as consequence:

$P(A \cap B) = P(A)\ P(B)$

Events A_1, A_2, …, A_n of a sample space S are statistically independent if and only if the joint probability of any combination of them is equal to the product of their respective marginal probabilities.

Random Variable

It is a function defined over the sample space, which transforms all the possible results of the sample space in numeric values.

A random variable is discrete if the quantity of values it can take is countable (finite or infinite) and if they can be arranged into a sequence that corresponds with the positive integer numbers.

Probability distribution of a discrete random variable

The domain of the random variable is the sample space. We can associate a probability measure to each possible random variable value.

Let X be a discrete random variable, and $\{x_j, j=1,2,3,...\}$ the set of the values X can take. The probability distribution of the random variable X is:

$P(X = x_j) = p(x_j) \geq 0$, j= 1, 2, 3, … and $\sum_j p(x_j) = 1$

Expected Value (or expectation) of a discrete random variable

The expected value of a discrete random variable X, denoted E(X), is the average or mean value of X.

$E(X) = \sum_x p(x)$

Mode of a discrete random variable

The mode of a discrete random variable X is the value x_m of X that maximizes the probability function, i.e. the most frequent value assumed by X.

Known probability distributions

There are specific probability distributions that have empirically demonstrated to be useful models for diverse practical problems. Their probability functions are mathematically deduced basing on certain hypothesis that are supposed to be valid for the considered random phenomenon.

Poisson Distribution

It is a discrete probability distribution where the random variable represents the number of independent events, each of which occurs with a known average rate and is independent of the time since the last event.

The distribution parameter is λ, the average number of occurrences of the random event in a time unit.

The Poisson process can be defined in three different (but equivalent) ways. We choose the following one:

The number of arrivals N(t) in a finite interval of length t obeys the Poisson(λt) distribution,

$$P\{N(t) = n\} = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

Moreover, the number of arrivals N(t1, t2) and N(t3, t4) in non-overlapping intervals ($t1 \leq t2 \leq t3 \leq t4$) are independent.

The expectation is: $E(N(t)) = \lambda t$

We can say that the resulting process of counting the number of events occurred in a time interval of duration t, has a Poissson distribution of parameter λt.

Descriptive Statistics

Statistics is the study of random phenomena. Its main aspect is the obtainment of conclusions based on experimental data (called statistic inference).

Descriptive statistics are used to describe the basic features of the data in a study. They provide simple summaries about the sample and the measures. They are useful when we have large data sets that can be considered as random samples. They can give the general distribution of values, basing on empirical evidence. Graphical description, tabular description and summary statistics are commonly used to summarize data. These techniques are based on the collection of data, its classification and the calculation of their relative frequency. The ***relative frequency*** is the quotient between the frequency of a class of observations and the total quantity of observations. The ***relative frequency histogram*** is the graphical representation of the relative frequencies of the classes.

Probability as relative frequency

As said before, one of the possible interpretations of probability is through relative frequency. The idea is that an experiment is carried out many times under the same conditions, and each time, a result is observed. The probability of the presence of certain attribute is approximated by the relative frequency of the results that have the attribute.

The definition of probability with this approach is the following:

> If an experiment is repeated *n* times under the same conditions and n_B of the results are favorable to an attribute B, the limit of n_B / *n* when *n* becomes large, is defined as the probability of the attribute B.

PASTA Property (Poisson Arrivals See Time Averages)

This property says that customers with Poisson arrivals see the system as if they came into the system at a random instant of time. The probability that when a customer arrives, he observes the system in a given state n, is equal to the probability that the system is in that state.

In our solutions we also apply some concepts of ***Statistical Reliability Theory*** [Gertsbakh-89], but we prefer to present them in the Section of their application for reasons of presentation clarity.

2.2 Application to our problem

In our problem we apply probability techniques to calculate the probability distributions of: (i) the quality values of the sources, (ii) the quality values provided by the DIS in certain data target, and (iii) the satisfaction of the user quality requirements by the DIS.

In all these cases, the *random experiment* considered is the observation of the DIS at a punctual moment. At this moment the sources have certain quality values, the DIS data target have certain quality values and the DIS is satisfying or not the user quality requirements.

For the experiment it is important to remark that we consider our observations have the *PASTA property*, ensuring there is not any correlation between the observations and the variation of the sources and system.

We define the *random variables* according to the model we are constructing. For example, in some cases we define the following ones:

X – Variable that represents the quality value of a source.

Y – Variable that represents the satisfaction or not of the user quality requirements by a source.

The *events* we define are sets of sources quality values combinations.

In all our models we *discretize* the quality values. The used criteria for this discretization vary according to the quality factor and the use of the quality value. When we discretize the values we must define two aspects: the precision we will manage for the values, and how we will round the values to the corresponding discrete value. The choice of the precision strongly depends on the particular case, for example, for freshness factor, the use of days, hours, minutes or seconds is determined by the characteristics of the data and data requirements in the DIS. The criteria we propose for rounding the values for freshness and accuracy are in general the following:

- In quality requirements we round the value to the most restrictive value. This implies that in the case of freshness we round it to the lowest value and in the case of accuracy to the highest one. Note that quality requirements are not only the ones directly posed by the users, but also the ones calculated in the system for being compared to the sources values (this is presented later in the document).
- In the sources quality models, in the case of freshness, values are rounded to the lowest value, since the idea is to transmit the quantity of time units that has effectively passed. In the case of accuracy, values are rounded to the nearest one.

3. Probabilistic Modeling of Sources Quality

We build probabilistic models where the random experiment consists on the verification of a source quality value, the random variable (RV) represents the source quality factor, and the sample space is the set of all its possible values. With this we have the probability distribution of the quality values at the source. The objective is to know the probabilities that hold for each of the possible quality values of the source if we query it at any moment. Having this distribution also gives us the possibility of calculating useful indicators such as expectation, mode, maximum and minimum.

In the case of freshness, for example, suppose we have the random variables X_1, X_2, …, X_n, so that each one corresponds to one of the n sources of the integration system. X_i represents the freshness value of source i at a given instant. The probability that $freshness_i = k_i$ is verified (where $freshness_i$ is the current freshness value of source i and k_i is a positive integer number), is: $p(X_i = k_i)$.

3.1 Models Construction

We build the models through three possible mechanisms, depending on the characteristics of the quality factor and the system, and on the available information.

Mechanism 1- Using an existing distribution. This can be done if the behavior of the quality factor or some property on which it is based, is already represented in a theoretical model.

Mechanism 2 - Calculating the distribution. This can be done if we have enough information about the behavior of the quality factor to deduce how the probabilities distribute across the possible values.

Mechanism 3 - Obtaining the probability distribution through the utilization of statistical techniques. We calculate the relative frequencies [Canavos-88] with the collected values of the quality factor. These relative frequencies, verifying some conditions depending on the case, are a good estimation of the respective probabilities [Canavos-88] [Cho+03].

In the cases of *Mechanism 1* and *Mechanism 2*, the calculation of the models may be aided by the use of SQL tools that specialize in extracting probabilistic models from data or generating known probabilistic models given the necessary information. One of these tools is SQLSAM, proposed in [Choobineh-95].

3.1.1 Freshness

In order to obtain the probability distribution of the freshness of a source, we need to know some specific characteristics of the source, which determine the mechanisms we can use. Following the definition of possible scenarios stated in Chapter 3, Section 2.4, we distinguish the different scenarios that are based on two of the dimensions, which are relevant to the source probabilistic modeling: (1) type of sources loading (periodically or continuously), and (2) meta-data provided by the sources (with respect to updates). The crossing of these dimensions generates the different scenarios.

In the following we show three probabilistic models we have developed for three different scenarios. The first one is built through Mechanism 2, the second through Mechanism 1 and the third through Mechanism 3.

Model 1

This model corresponds to a scenario where source loading is periodic and the available source meta-data is the update period. In this case we build the model calculating the distribution.

Consider a source S, let T be the period of the source loading, and X be the RV for the source freshness. The probabilities for the different possible values of freshness are all equal to 1/T:

$$p(x) \equiv p(X=x)$$
$$p(0) = p(1) = \ldots = p(T-1) = 1/T$$

We also calculate the expectation for X, which coincides with the average of the possible freshness values:

$$E(X) = \Sigma_x \, xp(x) = 0(1/T) + 1(1/T) + ... + (T-1)(1/T) = (0+1+\ldots+T-1) / T$$

On the other hand, supposing we start at an instant $t_0=0$, we have an expression for the freshness at a given instant t:

$$freshness(t) = t - (t \text{ DIV } T) * T$$

Figure 4.1 shows a graph that represents the variation of freshness through time, in the case of T=5.

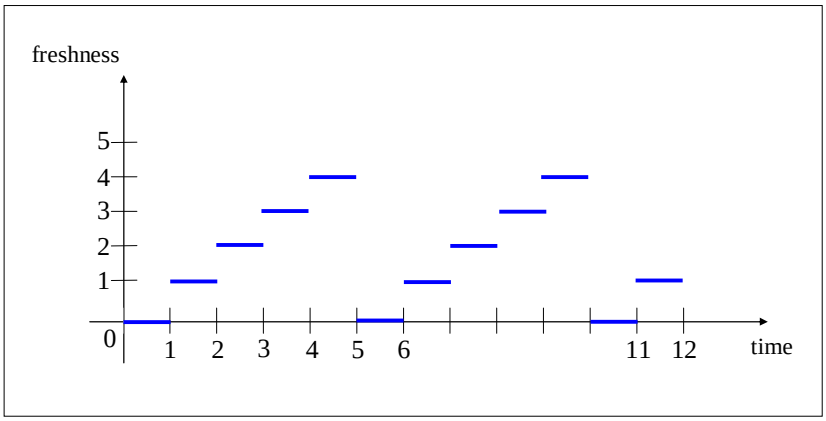


Figure 4.1: Freshness values through time

Examples of sources that are updated periodically are:

- Salaries system database. A database that is updated each 30 days, when salaries are liquidated.
- Cinema billboard. A cinema where the billboard is updated once a week.
- Data Warehouse. A data warehouse that is updated each 24 hours.

Model 2

In this model the source loading is in a continuous basis. We assume that source updates follow a Poisson process and that the update frequency λ is available. We chose this example taking into account the paper [Cho+03], where the authors assume that a source element changes by a Poisson process, mentioning experimental data that shows this behavior for web pages. With respect to λ estimation, they say that in the case of "complete history of changes", it is well known that (number of changes)/(monitoring period) gives "good" estimation of the frequency of change.

We build the model deducing the probability distribution of freshness values from the probability distribution of the quantity of updates in the source, which is a Poisson distribution.

Given a source S, the RV X represents the quantity of updates in a time unit, and the RV Y represents the source freshness. We know the distribution of X, we deduce the distribution of Y.

The probability that there is at least one update in a certain time interval (p_U) is the complement of the probability that there is zero updates in this time interval:

$$p_U = 1 - p(X=0)$$

The probability that the freshness at the end of certain time interval is 0, is equal to p_U. The probability that the freshness is 1, is equal to the probability that there has been an update in the previous time interval, multiplied by p_U. In this way we can obtain the distribution for the RV Y:

$$p(Y=0) = p_U$$
$$p(Y=1) = p_U.p(X=0)$$
$$p(Y=2) = p_U.p(X=0).p(X=0)$$
……

We also calculate the expectation:

$$E(Y) = \Sigma_y \, yp(y) = p(X=0) + (p(X=0))^2 + ...$$

In addition, this model is a stochastic process and also a Markov chain [Hillier+91] [IOCourse-07].

A stochastic process $X = (X_t, t \in T)$ is a family of random variables that describe the evolution of an experiment, through time. $t \in T$ is the time parameter of process X. Each possible value of RV X_t is a possible system state. $X_t = e$ indicates that at instant t the process is in state e.

$\{X_i\}$ is a stochastic process, where X_i is the quantity of updates in each time interval. The X_i are independent RVs and identically distributed with known probability distribution, Poisson.

$\{Y_i\}$ is a stochastic process, where Y_i is the freshness at the end of i interval. The Y_i are dependent RVs and they can be evaluated iteratively through the expression:

$$Y_{t+1} = \begin{cases} Y_t + 1, & \text{if } X_{t+1} = 0 \\ 0, & \text{if } X_{t+1} \geq 1 \end{cases}$$

$$Y_0 = 0$$

The $\{Y_i\}$ stochastic process has the markovian property, which roughly means that given the present, the future is conditionally independent of the past:

$$p\{Y_{t+1} = j \mid Y_0 = k_0, \ldots, Y_{t-1} = k_{t-1}, Y_t = i\} = p\{Y_{t+1} = j \mid Y_t = i\}$$

Additionally, the transition probabilities $p\{Y_{t+1} = j \mid Y_t = i\}$ are stationary, since:

$$p_{ij} = p\{Y_{t+1} = j \mid Y_t = i\} = p\{Y_1 = j \mid Y_0 = i\} \qquad \text{for all } t = 0, 1, \ldots$$

This characteristic gives us the possibility of calculating the transition matrix, obtaining all the transition probabilities, i.e. the probabilities that the freshness changes from certain value to another.

The following is an example where this model is applied.

Example 4.1

The considered data source is a database table of a bank. In the bank the clients arrive following a Poisson process and also the database updates occur according to this process. The estimated update frequency for the table is 1 update each 4 minutes.

We choose the interval length as 4 minutes. When using this method the time interval we choose for the Poisson model of the updates is necessarily the time unit the model for freshness will have. Therefore, our values for freshness will be multiples of 4 minutes.

t = 4, time interval (minutes)

$\lambda = 1$, estimated quantity of updates in 4 minutes

Let X be the RV that represents the quantity of updates of the source, and Y the RV that represents the freshness of the source measured in time units (of 4 minutes).

$P(X=0) = (\lambda^n/n!)e^{-\lambda} = (\, 1^0/0! \,)\, e^{-1} = e^{-1} = 1/e^1 = 0.36$ -- Probability of 0 updates in a 4-minutes interval

$p_U = 1 - p(X=0) = 1 - 0.36 = 0.64$ -- Probability that freshness is equal to 0 at the end of the interval

The distribution of RV Y is the following:

$p(Y=0) = p_U = 0.64$
$p(Y=1) = p_U.p(X=0) = 0.64 \times 0.36 = 0.2304$
$p(Y=2) = p_U.p(X=0).p(X=0) = 0.64 \times (0.36)^2 = 0.082944$
$p(Y=3) = p_U.p(X=0).p(X=0).p(X=0) = 0.64 \times (0.36)^3 = 0.02985984$
$p(Y=4) = p_U.p(X=0).p(X=0).p(X=0).p(X=0) = 0.64 \times (0.36)^4 = 0.0107495424$
$p(Y=5) = p_U.p(X=0).p(X=0).p(X=0).p(X=0).p(X=0) = 0.64 \times (0.36)^5 = 0.003869835264$
…….
$p(Y=n) = p_U.(p(X=0))^n = 0.64 \times (0.36)^n$

Intuitively, we can see that the deduced distribution for freshness is correct. See the graph presented in Figure 4.2. This is the graph of the probability distribution (Poisson distribution) of the quantity of updates at the source. It shows that the highest probability is for the value 1, and the values near 1 have also a high probability. As the values increase, the probability is drastically lower. The value 0 for freshness has a high probability (P(Y=0)), because this probability is equal to the sum of the probabilities corresponding to quantity of updates greater than 0, which is the probability $P(X \geq 1)$. The probabilities corresponding to freshness values greater than 0 ($P(Y=v)$, $v>0$), are noticeable lower because they are equal to the product of several probabilities of X; the greater is the value, the greater is the quantity of times we multiply by P(X=0), which is equal to 0.36.

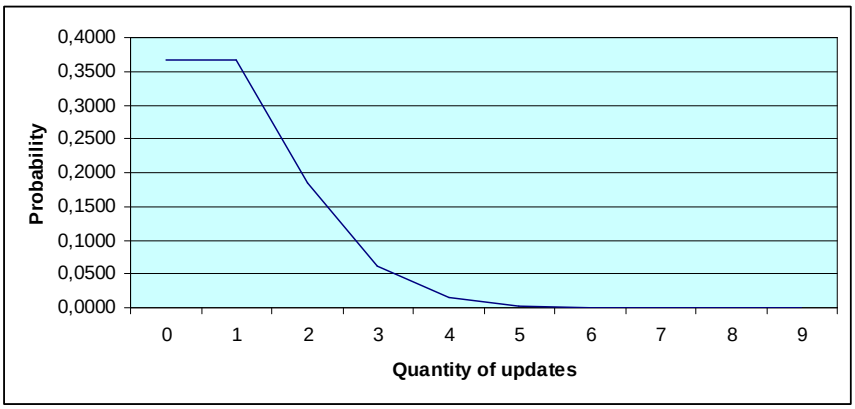


Figure 4.2: Poisson Distribution of Quantity of Updates

We also calculate the expectation for freshness values:

$E(Y) = \Sigma_y\, yp(y) = \Sigma_y\, (y \times 0.64 \times 0.36^y) = 0.36 \times 0.64 \times \Sigma_y\, y \times 0.36^{y-1} =^1 0.5625$

Finally, we can express freshness **in terms of minutes**. We consider a RV Z that represents the freshness of the source, measured in minutes, having the following distribution:

p(Z=0) = 0.64	p(Z=8) = 0.082944	p(Z=16) = 0.0107495424
p(Z=4) = 0.2304	p(Z=12) = 0.02985984	p(Z=20) = 0.003869835264

Expectation now is: E(Z) = 2.25
◊

[1] We apply the following equality: $\Sigma_n\, n\, p^{n-1} = 1 / (1 - p)^2$, where $p < 1$

Model 3

In this model the source loading is in a continuous basis. The information we have for constructing the model is an events repository where each event is generated when a data update occurs at the source. The following are the steps for obtaining the model:

- From the events repository calculate a table *T*, which will be the sample for our model, where:
  - there is a tuple for each "time point", according to the defined timestep
  - each tuple has a timestamp (the time point)
  - each tuple has the freshness value corresponding to the timestamp
- Considering the last *n* tuples inserted in table *T*, construct a relative frequencies table, containing for each freshness value that occurs in *T*, an index that is calculated as the quantity of occurrences in *T* of the freshness value, divided by *n*.
- The probability distribution is derived from the relative frequencies, assigning a probability value for each possible freshness value as follows:
  - For the freshness values that appears in the relative frequencies table, the probability is equal to the corresponding index, rounded to a predefined precision
  - For the freshness values that do not appear in the relative frequencies table, the probability is equal to 0

The number *n* of tuples considered as enough to build the relative frequencies table (our sample) must be determined according to the context, as well as the precision of the probability values. From the relative frequencies table, a frequency histogram can be constructed, which shows a graphic description of the data.

We show an example where this model is constructed.

Example 4.2

Consider a source relation that stores data about meteorological predictions, which comes from a satellite in real time. The source is updated irregularly.

Table 4.1 is a portion of the events repository, where Date-time is the date and time in the format "dd/mm-hh", where time is represented in hours.

Source	**Event**	**Date-time**
S1.Meteo	data update	2/11-2
S1.Meteo	data update	2/11-3
S1.Meteo	data update	2/11-5
S1.Meteo	data update	2/11-8
S1.Meteo	data update	2/11-9
S1.Meteo	data update	2/11-10
S1.Meteo	data update	2/11-12
S1.Meteo	data update	2/11-14
S1.Meteo	data update	2/11-18
S1.Meteo	data update	2/11-21
S1.Meteo	data update	2/11-22

Table 4.1: Events Repository

From the events repository we calculate the freshness that held at each time point (see Table 4.2), considering the first time point as 2/11-2, the first timestamp of the events repository, and a time step of 1 hour.

Source	Time point	Freshness
S1.Meteo	2/11-2	0
S1.Meteo	2/11-3	0
S1.Meteo	2/11-4	1
S1.Meteo	2/11-5	0
S1.Meteo	2/11-6	1
S1.Meteo	2/11-7	2
S1.Meteo	2/11-8	0
S1.Meteo	2/11-9	0
S1.Meteo	2/11-10	0
S1.Meteo	2/11-11	1
S1.Meteo	2/11-12	0
S1.Meteo	2/11-13	1
S1.Meteo	2/11-14	0
S1.Meteo	2/11-15	1
S1.Meteo	2/11-16	2
S1.Meteo	2/11-17	3
S1.Meteo	2/11-18	0
S1.Meteo	2/11-19	1
S1.Meteo	2/11-20	2
S1.Meteo	2/11-21	0
S1.Meteo	2/11-22	0

Table 4.2: Calculated freshness

Then we calculate the relative frequencies of the freshness values (see Table 4.3). For each freshness value, the relative frequency is the number of occurrences of the value divided by 21, which is the quantity of hours chosen as our sample.

Source	Freshness value	Relative frequency
S1.Meteo	0	11/21
S1.Meteo	1	6/21
S1.Meteo	2	3/21
S1.Meteo	3	1/21

Table 4.3: Relative frequencies of freshness values

Let X be the RV that represents the freshness of source S1.Meteo. The following is the probability distribution of X:

$P(X=0) = 0.52$
$P(X=1) = 0.29$
$P(X=2) = 0.14$
$P(X=3) = 0.05$

We calculate the expectation of X as follows:

$$E(X) = \Sigma_x \, xp(x) = 0*0.52 + 1*0.29 + 2*0.14 + 3*0.05 = 0.72$$

◊

3.1.2 Accuracy

To obtain the probability distribution of the accuracy quality factor of a source we apply the mechanism of calculating the relative frequencies with the collected values (Mechanism 3). For this we need to have a periodic measurement of the accuracy of the source. The method we propose is the following:

- Source accuracy is measured each *n* time-units.
- When *m* accuracy measurements have been done, a relative frequencies table is constructed, containing for each possible accuracy value, an index that is calculated as the quantity of occurrences of the value divided by *m* (the quantity of measurements done).
- We consider these relative frequencies as an estimation of the probability distribution of the accuracy values.

We show this through the following example.

Example 4.3

Suppose we have a source table that stores meteorological information and is updated three times a day by humans operators. The accuracy of this table is measured twice a week. Table 4.4 shows the measured values.

Source-Table	Measure	Date
S1-Temperature	0,6	7/3/07
S1-Temperature	0,5	10/3/07
S1-Temperature	0,6	14/3/07
S1-Temperature	0,6	17/3/07
S1-Temperature	0,8	21/3/07
S1-Temperature	0,7	24/3/07
S1-Temperature	0,7	28/3/07
S1-Temperature	0,8	31/3/07
S1-Temperature	0,6	4/4/07
S1-Temperature	0,7	7/4/07
S1-Temperature	0,7	11/4/07
S1-Temperature	0,7	14/4/07
S1-Temperature	0,7	18/4/07
S1-Temperature	0,8	21/4/07
S1-Temperature	0,7	25/4/07
S1-Temperature	0,7	28/4/07

Table 4.4: Accuracy measurements.

We construct the relative frequencies table and each two months we update it. Table 4.5 is the relative frequencies table that corresponds to the data in Table 4.4.

Source-Table	Acc-value	Relative frequency
S1-Temperature	0	0
S1-Temperature	0.1	0
S1-Temperature	0.2	0
S1-Temperature	0.3	0

S1-Temperature	0.4	0
S1-Temperature	0.5	1/16
S1-Temperature	0.6	4/16
S1-Temperature	0.7	8/16
S1-Temperature	0.8	3/16
S1-Temperature	0.9	0
S1-Temperature	1	0

Table 4.5: Relative frequencies.

We consider this table as an estimation of the probability distribution of accuracy values in table Temperature of S1. Considering the RV X as the accuracy, the rounded probability values are:

$p(X=0.6) = 0.3$
$p(X=0.7) = 0.5$
$p(X=0.8) = 0.2$

The expectation for X is calculated. It coincides with the average of the measured values.

$$E(X) = \Sigma_x \, xp(x) = 0.6*0.3+0.7*0.5+0.8*0.2 = 0.69 \approx 0.7$$

◊

3.2 Models Specification

We specify a source quality model through the probability distribution of its quality values. For each source and quality factor there is a set of value-pairs that have the probability associated to each possible quality value.

Definition 4.1: A *source quality behavior model* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, source, distribution>, where

- qfactor is a String, representing the quality factor,
- source is a String, the name of the source to which the model corresponds,
- distribution is a set of pairs <qvalue, probability>, where
  - qvalue is a Decimal number and
  - probability is a number between 0 and 1, representing the probability of the quality value.

□

This information allows obtaining, for a source and a quality factor, besides the probability associated to each value, the expectation, the mode, and the maximum and minimum values.

We include sources quality behavior models in the *quality graph* (presented in Chapter 3), each one as a *property* associated to a source.

4. Probabilistic Modeling of System Quality

In order to model DIS quality we first must define what the quality of the DIS is for us. As we previously said, quality is better, the nearer its values are to the quality values required by the user. Therefore DIS quality not only involves the quality values provided by the DIS to the user, but also the satisfaction or not of the user quality requirements. DIS quality is affected by sources quality,

transformations that are applied to sources data, and user quality requirements. To model it we must take into account the three elements.

The DIS model may be given through three different aspects: (1) The *DIS Quality Certainty*, which is the probability that the DIS satisfies the quality requirements (those whose type is "probability"), (2) the information about the satisfaction of the requirements of maximum, minimum, average and most frequent value, and (3) the probability distribution of the possible quality values satisfied by the DIS. Calculation of (1) is done when there is at least one requirement of type "probability", i.e. that gives a condition and a probability for its satisfaction. Calculation of (2) only has sense if there are any requirements of type "maximum", "minimum", "average" or "frequency". The three aspects can be calculated and used simultaneously.

Different approaches could be chosen to calculate (1) and (2). We choose to propagate the user quality requirements to the sources and verify the satisfaction of these requirements by the sources' models. In the propagation we take into account the transformations that are applied to the data. We emphasize in the satisfaction of quality requirements *working at sources level*, which gives us more information about each source influence in DIS quality, especially useful at change detection-and-management time. We call *accepted configurations* to the requirements at sources' level.

In the calculation of (3) user quality requirements are not taken into account. The possible sources quality values are propagated to the data targets in order to obtain the DIS quality distribution.

In the following sub-sections we present the definition and calculation of *accepted configurations*, the calculation of *DIS quality certainty* and of the satisfaction of "non-probability" requirements, and the probability distribution of the quality provided by the DIS.

4.1 Accepted Configurations

Given a quality graph and a set of quality requirements associated to its target nodes, the *accepted configurations* are all the combinations of quality values, each value associated to a source, that make the quality requirements satisfied. The quality requirements considered for this calculation must be all of the same type, for example we calculate accepted configurations for requirements of average or for requirements of maximum, etc.

From the user quality requirements we deduce the quality restrictions the sources must satisfy, existing different combinations of sources quality values that satisfy these restrictions, which are the accepted configurations.

As said before, we need the accepted configurations to model the quality of the DIS, since this quality depends on how the system achieves the satisfaction of the quality requirements. However, the accepted configurations are also very useful from other viewpoints. For example, each time there is a source quality change it is not necessary to re-calculate the system quality to know if it satisfies the requirements. On the other hand, they may be used as an assistance for negotiation with sources (if you want to ask another source to improve some quality value).

The following are the concepts we manage for representing the accepted configurations:

Definition 4.2: A *source restriction* is a restriction to be verified by the quality factor of the source. It is a 4-uple r = <qfactor, source, op, value>, where:

- qfactor is a String, representing the quality factor
- source is a source node of the quality graph
- op $\in$ {"<", "≤", ">", "≥", "="}, is a comparison operator

- value is a Decimal □

Definition 4.3: A *restriction vector* is a set of source restrictions, one restriction for each one of the DIS sources, where all the restriction operators are the same. We use the following notation:

$\nu = < r_{S1}, \ldots, r_{Sn}>$, where r_{Si} is the restriction associated to source S_i □

Definition 4.4: A *restriction-vector space* is a set of restriction vectors. We use the following notation:

$\aleph = \{\nu_1, \ldots, \nu_m\}$, where ν_i is a restriction vector. □

We specify the accepted configurations as a property, which is associated to the graph. It may exist many different accepted configurations associated to the quality graph.

Definition 4.5: The *accepted configurations* is a property type, whose corresponding domain is a set of pairs of the form: <req-set, rv-space>, where:

- reqs-set is a set of pairs of the form <target, req-name>, where:
  - target is the data target to which the requirement is associated
  - req-name is a String (the name of the requirement)
- rv-space is a restriction-vector space, which corresponds to the requirements of reqs-set. □

Note that the accepted configurations contain the solution space for the satisfaction of a set of user quality requirements. If the sources satisfy any of the restriction vectors of the solution space, they are satisfying these requirements.

We calculate the accepted configurations by means of the propagation of the user quality requirements to the sources. For each target node requirement (user quality requirement), we traverse the quality graph from the target node to the source nodes, starting with an equation (it may be an equation or an inequation, depending on the type of requirement, but for simplicity we use the word "equation") whose variable is the quality value of the activity node that is predecessor of the target[1], and finally obtaining an equation, whose variables are the sources quality values. This equation is the restriction the sources quality values must satisfy in order to satisfy the user quality requirement considered. When we have processed all target node requirements, we obtain an equation system, whose variables are the sources quality values.

We present a general algorithm to achieve this calculation and associating it to the quality graph. Our algorithm has as precondition that the propagation for evaluating the quality of the graph target nodes is pre-defined. This means that we know how the quality values of each activity node are calculated in function of its predecessors. We assume we have a function, called **Propagation-expression,** that returns the calculation expression for an activity node. This problem is studied and solutions are proposed for freshness and accuracy in [Peralta-06].

In the following we present a pseudo-code of the algorithm, which intends to describe the steps we follow to calculate the accepted configurations for a graph, a set of requirements and one quality factor. This is a general approach, since the specific implementations are different according to the quality factor considered and the specific characteristics of the DIS.

[1] According to [Peralta-06], target nodes have a unique incoming data edge.

```
TYPE
    TargetRequirement: target: DataTarget
                       req-name: String

    RequirementExpression:   op: Operator
                             value: Decimal
    Operator: (<, ≤, >, ≥, =)
    Equation:  exp: Expression
               op: Operator
               dec: Decimal
    Expression: String
    EquationSystem: Set of Equation


FUNCTION Accepted_Conf_Calculation (G: QualityGraph, qfactor: String, reqs: Set of
                                    TargetRequirement): AcceptedConfigurations

R: TargetRequirement
requirement: UserQualityRequirement
reqexp: RequirementExpression
equ, source-equ, e: Equation
r: SourceRestriction
equation-system: EquationSystem
y: RestrictionVectorSpace

Create(y)
FOR EACH R of reqs DO
    requirement = Obtain_requirement (R)
    reqexp = Obtain_requirement_expression (requirement)
    equ = Generate_satisfaction_equation (G, R.target, reqexp)
    source-equ = Obtain_sources_equation (G, equ)
    IF it is possible to obtain a set of equations, each one depending on only 1 source
THEN
        FOR EACH obtained equation e corresponding to source S DO
            IF Exists_source_restriction (y, S) THEN
                r = Obtain_source_restriction (y, S)
                IF r is less restrictive than e THEN
                    Change_source_restriction (y, e, r, S, qfactor)
            ELSE
                Add_source_restriction (y, e, S, qfactor)
    ELSE
        Insert (source-equ, equation-system)
IF equation-system is not empty THEN
    y= Obtain_solutions (equation-system)
Accepted_Conf_Calculation = <reqs, y>

END FUNCTION
```

The following are the descriptions of the used functions and procedures:

- **Obtain_requirement** (*R*: TargetRequirement): UserQualityRequirement

  It obtains the requirement: ρ_V (*R.target*) ("UserQualityRequirement") (*R.req-name*)

- **Obtain_requirement_expression** (*req*: UserQualityRequirement): RequirementExpression

  It obtains from a user quality requirement a pair *(op, value)* according to the requirement's type and quality factor. E.g.: (≤, 20)

- **Generate_satisfaction_equation** (*G*: QualityGraph, *t*: Node, *reqexp*: RequirementExpression): Equation

  It generates a 3-uple <*last-activity*, *op*, *value*>, where: *last-activity* is the activity node that is predecessor of *t*, and *op* and *value* are extracted from *reqexp.*

- **Obtain_sources_equation** (*G*: QualityGraph, *equ*: Equation): Equation

  It generates an equation through the following steps:
  expression = **Propagation-expression** (*equ.exp*)
  WHILE there is an activity node *A* contained in *expression* DO
  // It ends when all nodes in *expression* are source nodes.
  expression = Substitute *A* by Propagation-expression (*A*)
  Build equation as: <*expression, op, value*>

- **Propagation-expression (***activity-name*: String): Expression

  Given an activity node, it returns the expression that corresponds to the calculation of the quality factor value in that activity node.

- **Exists_source_restriction** ($\mathcal{Y}$: RestrictionVectorSpace, *S*: Node): Boolean

  It verifies if:
  - it exists a restriction vector $v \in \mathcal{Y}$ and
  - it exists a source restriction $r \in v$, where r.source = *S*

- **Obtain_source_restriction** ($\mathcal{Y}$: RestrictionVectorSpace, *S*: Node): SourceRestriction

  It returns the source restriction r that verifies:
  - r.source = *S*
  - $r \in v$, v is a restriction vector of $\mathcal{Y}$

- **Change_source_restriction** ($\mathcal{Y}$: RestrictionVectorSpace, *e*: Equation, r: SourceRestriction, *S*: Node, *qfactor*: String)

  It substitutes in $\mathcal{Y}$ the source restriction r by the source restriction <*qfactor*, *S*, *e.op*, *e.value* >

- **Add_source_restriction** ($\mathcal{Y}$: RestrictionVectorSpace, *e*: Equation, *S*: Node, *qfactor*: String)

  It adds the source restriction <*qfactor, S, e.op, e.value* > to $\mathcal{Y}$

- **Insert** (*equ*: Equation, *equation-system*: EquationSystem)

  It inserts an equation into a set of equations.

- **Obtain_solutions** (*equation-system*: EquationSystem): RestrictionVectorSpace

  It returns the set of restriction vectors that satisfy all the equations of the equations system.

Note that this algorithm calculates the accepted configurations for one quality factor and a set of user requirements that are posed for this quality factor. As said before, the user quality requirements of this set must be all of the same type, otherwise the algorithm may not work correctly. The different quality requirements involve different sources, according to the sub-graph that generates the data target corresponding to the quality requirement.

We distinguish two possible cases for the calculated accepted configurations. They depend on the way the quality values are propagated from the sources to the target, more precisely; in the way the quality is calculated in a node in function of its predecessors (we call it "propagation function"). These cases are the following:

1- The accepted configurations consist of an only one restriction-vector. This means that there is a fixed restriction for each source relation, for example: "freshness(Source2)<15". This occurs when in the propagation of quality requirements from targets to sources, the generated equation can be decomposed in various new equations such that each new equation has only one variable (a source quality value). This is shown more clearly in Example 4.4 presented below.

2- The accepted configurations consist of a set of restriction vectors (a vector space). This means that there is a set of possible combinations of sources quality values, where each combination satisfies the user quality requirements. This occurs when in the propagation of quality requirements from targets to sources, for each requirement an equation is generated that has several variables (representing the source quality values).

Observing the algorithm, it can be noted that each processed quality requirement can generate: (i) new source restrictions, which substitute the existing source restrictions, or (ii) new restrictions that combine several sources and are combined with other restrictions.

In order to experiment this proposal we worked with a study case. We considered a system that integrates information from several hospitals of a country (presented as example in Chapter 1). In the following we present as an example some of the calculation of the accepted configurations for the quality factor freshness in the quality graph of our case study.

Example 4.4

Figure 4.4 shows the quality graph of the DIS of the study case, which works with five source relations. The activity nodes have labels that represent the cost of the activity expressed in hours. The target nodes have labels expressing the required values for freshness, giving the maximum quantity of hours they accept. The edges have labels that represent a synchronization delay between one activity and the other, expressed in hours.

The freshness propagation function in this case is the following for any activity node A:

$Freshness(A) = Max\ (\ Freshness(A_1) + Delay(A_1,A),\ \ldots,\ Freshness(A_n) + Delay(A_n,A)\ ) + Cost(A)$, where $A_1, \ldots, A_n$ are A predecessors

We calculate the accepted configurations:

1) We propagate requirement of T1:

   We generate the first inequation in function of the predecessor of T1.

   fr(A6) ≤ 24

   We substitute fr(A6) by an expression in function of sources freshness.

   Max (fr(A1) + 3.5, fr(A2) + 3.5, fr(A5) + 0) + 0 ≤ 24
   fr(A1) = fr(S1) + 0 + 0.5

fr(A2) = Max (fr(S2) + 0, fr(S4) + 0) + 0.5
fr(A5) = fr(A4) + 0 + 1
Max (fr(S1) + 0.5 + 3.5, Max (fr(S2), fr(S4)) + 0.5 + 3.5, fr(A4) + 1) ≤ 24
fr(A4) = fr(S5) + 0 + 3
Max (fr(S1) + 0.5 + 3.5, Max (fr(S2), fr(S4)) + 0.5 + 3.5, fr(S5) + 3 + 1) ≤ 24

We obtain several inequations with only one variable.

fr(S1) + 0.5 + 3.5 ≤ 24
Max (fr(S2), fr(S4)) + 0.5 + 3.5 ≤ 24
fr(S2) + 0.5 + 3.5 ≤ 24
fr(S4) + 0.5 + 3.5 ≤ 24
fr(S5) + 3 + 1 ≤ 24

Source restrictions: **fr(S1) ≤ 20**
fr(S2) ≤ 20
fr(S4) ≤ 20
fr(S5) ≤ 20

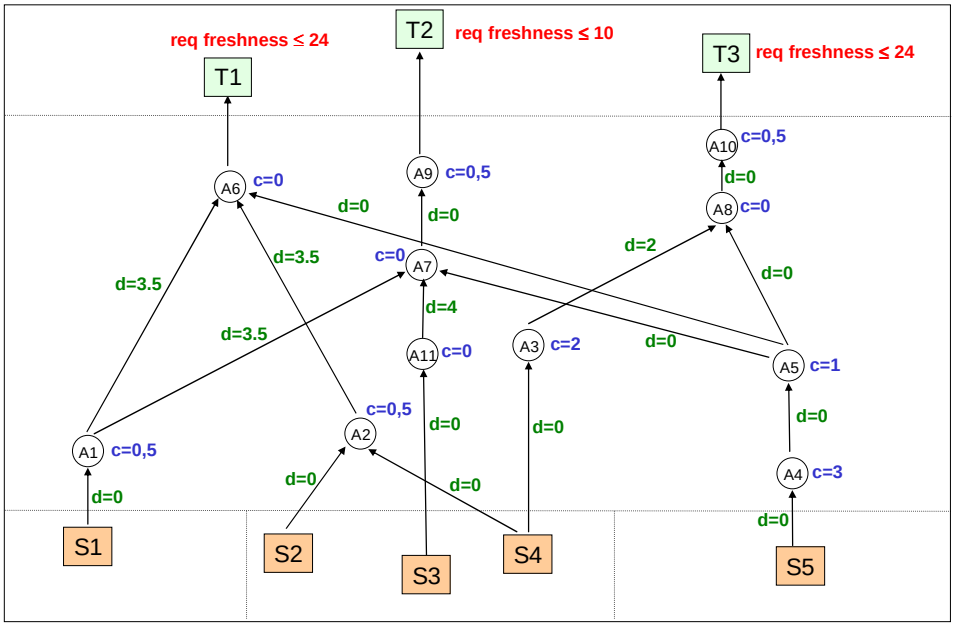


Figure 4.4: Quality graph of Study Case

2) We propagate requirement of T2:

We generate the first inequation in function of the predecessor of T2.

fr(A9) ≤ 10

Analogously to 1), we obtain:

Source restrictions: **fr(S1) ≤ 5.5**
fr(S3) ≤ 5.5
fr(S5) ≤ 5.5

3) We propagate requirement of T3:

We generate the first inequation in function of the predecessor of T3.

fr(A10) ≤ 24

Analogously to 1), we obtain:

Source restrictions: **fr(S4) ≤ 19.5**
fr(S5) ≤ 19.5

The accepted configurations are:

fr(S1) ≤ 5.5
fr(S2) ≤ 20
fr(S3) ≤ 5.5
fr(S4) ≤ 19.5
fr(S5) ≤ 5.5

◊

In [Peralta-06] there is a proposal for freshness requirements propagation in a specific scenario.

4.2 Satisfaction of Probabilistic User Quality Requirements

Probabilistic user quality requirements are the requirements (presented in Chapter 3, Section 3) that express a value and an associated probability (of type "probability"). The meaning is that the value must be verified (as maximum or minimum) with the given probability.

Using the sources quality models and the accepted configurations, we calculate with which probability the DIS can verify the values stated in all the probabilistic requirements. For this, we first model the satisfaction of the accepted configurations by the sources, and then we calculate the DIS Quality Certainty.

4.2.1 Probabilistic Modeling of Accepted Configurations Satisfaction

In Section 3 we presented how we model the quality of the sources. Basing on those models we calculate the model of the satisfaction of accepted configurations by the sources.

Before working with the combination of the sources models and the accepted configurations, we need to make the values managed in both cases, compatible. This means that the values of the accepted configurations and the possible values of the sources must be in the same units and with the same precision. We must consider that in the sources models we discretize the possible values of the sources quality, therefore, in order to allow the satisfaction of the restrictions, the values used in the restrictions must be discretized with the same discretization unit. Furthermore, to discretize the values of the restrictions we must take into account the comparison operator used. If it is "<" or "≤" the value must be rounded down, while if it is ">" or "≥" it must be rounded up. If the operator is "=" we must take into account the quality factor to decide how to round the value. In the following we show an example of the discretization of the restrictions values, and how they are compared to the sources values.

Example 4.5

Continuing with Example 4.4, we convert the obtained accepted configurations as follows:

fr(S1) ≤ 5.5 → **fr(S1) ≤ 5**
fr(S2) ≤ 20

fr(S3) ≤ 5.5 → **fr(S3) ≤ 5**
fr(S4) ≤ 19.5 → **fr(S4) ≤ 19**
fr(S5) ≤ 5.5 → **fr(S5) ≤ 5**

Note that if S1 has a freshness value of **5.8**, in the source model it is considered as **5** (the idea is that 5 hours has passed, not 6), so the restriction is verified. However, if the restriction was **fr(S1) < 5.5**, it would be converted to **fr(S1) < 5**, which would not be verified by S1.

If we have, for accuracy, a restriction:

acc(S3) ≥ 0.73, we must convert it to **acc(S3) ≥ 0.8.**

Note that if S3 has an accuracy value of **0.74**, it is discretized to **0.7**, and the restriction is not verified. However, if S3 has an accuracy value of **0.76**, it is descretized to **0.8**, and the restriction is verified.

If we have the following restriction:

fr(S5) = 5.5, we convert it to **fr(S5) = 5**, such that we prevent the passing of freshness values greater than 5.5

If we have the restriction:

acc(S3) = 0.73, we convert it to **acc(S3) = 0.8**, such that we prevent the passing of accuracy values lower than 0.73

◊

When modeling the quality of a source, the random experiment consists on the verification of a source quality value, the random variable (RV) X represents the source quality factor, and the sample space is the set of all its possible values. In order to model the accepted configurations satisfaction by a source, we define a binary RV Y, whose possible values are 0 and 1. Y maps the values of the sample space to these two values. We define Y in function of X and of the accepted configurations:

Definition 4.6:

Let $\mathcal{Y}$ be the accepted configurations, $\mathcal{Y} = \{\nu_1, \ldots, \nu_m\}$, where $\nu_i = <r_{iS1}, \ldots, r_{iSn}>$, and r_{iSj} is the restriction associated to source S_j in the restriction vector ν_i, $r_{iSj} = <qfactor, S_j, op, value>$

Let X be the RV that represents the quality factor *qfactor* in source S_j, over the sample space $\mathcal{S}$

Let $(x_1, x_2, \ldots, x_h)$ the values of $\mathcal{S} / x_i \in$ Real and $x_i < x_{i+1}$ and $(\exists\, k / x_k = value)$

Y is a random variable over $\mathcal{S}$, such that:

For restriction vector ν_i and source S_j, Y = 1 if (X *op value*)

Y = 0 if not (X *op value*)

□

The probability distribution of Y is calculated in function of the probability distribution of X. This calculation depends on the *op* comparison operator, according to these cases:

1) *op* = "<"

   $P(Y = 1) = P(X < value) = \sum_{i=1..k-1} P(X=x_i)$

2) *op* = "≤"

   $P(Y = 1) = P(X \leq value) = \sum_{i=1..k} P(X=x_i)$

3) *op* = ">"

   $P(Y = 1) = P(X > value) = \sum_{i=k+1..h} P(X=x_i)$

4) *op* = "≥"

   $P(Y = 1) = P(X \geq value) = \sum_{i=k..h} P(X=x_i)$

5) *op* = "="

   $P(Y = 1) = P(X = value) = P(X=x_k)$

For all cases, $P(Y = 0) = 1 - P(Y = 1)$

In the definition of Y, for simplicity, we considered that the sample space is discrete and finite. However there are some cases where we need to consider a sample space discrete and infinite. For example, this may occur in the case of freshness, when it is not possible to anticipate which value the freshness can reach. In these cases, the values of the sample space S are (x_1, x_2, …) and some probabilities must be calculated as the complement of others:

If *op* = ">"

$P(Y = 1) = P(X > value) = 1 - P(X \leq value) = 1 - \sum_{i=1..k} P(X=x_i)$

If *op* = "≥"

$P(Y = 1) = P(X \geq value) = 1 - P(X < value) = 1 - \sum_{i=1..k-1} P(X=x_i)$

In the following we show an example of a model of accepted configurations satisfaction.

Example 4.6

We retake the Example 4.2, which is a source updated irregularly (S1.Meteo), whose data is about meteorological predictions and comes from a satellite in real time.

X is the RV that represents the freshness of S1.Meteo and the following is its probability distribution:

P(X=0) = 0.52
P(X=1) = 0.29
P(X=2) = 0.14
P(X=3) = 0.05

The accepted configurations has for source S1.Meteo, the restriction: fr(S1.Meteo) ≤ 2

Let Y be the RV that represents the accepted configurations satisfaction by S1.Meteo. We calculate distribution of Y:

P(Y = 1) = P(X ≤ 2) = P(X=0) + P(X=1) + P(X=2) = 0.52 + 0.29 + 0.14 = 0.95

P(Y = 0) = 1 – P(Y = 1) = 0.05

Therefore, there is a probability of 0.95 that S1.Meteo verifies the accepted configurations.

◊

We model the satisfaction of one restriction by one source through RV Y. To model the satisfaction of a restriction vector $v = <r_1, \ldots, r_n>$ by the n sources involved, it is necessary to have the distributions of the RVs $Y_1, \ldots, Y_n$ / Y_i represents the satisfaction of r_i by source S_i. To obtain this we must have the distributions of $X_1, \ldots, X_n$ / X_i is the RV that represents the quality factor values of source S_i.

4.2.2 DIS Quality Certainty

Considering the requirements of type "probabilistic", we define DIS Quality Certainty as follows:

Definition 4.7: *DIS Quality Certainty* is the probability that the DIS quality requirements are satisfied. □

This concept and its calculation were inspired by Reliability Theory [Gertsbakh-89], in particular some of its techniques.

According to Reliability Theory, "the word reliability refers to the ability of a system to perform its stated purpose adequately ... under the operational conditions encountered". In particular, *structural reliability* relates the reliability of the components of a system and the reliability of the whole system. It is based on the *structure function*, which relates the state of the system to the state of its components. The possible states are two, operational (up) and failure (down). The state of component i, i= 1, 2, ..., n, is described by a binary variable X_i / X_i =1 if the component is up and X_i = 0 if the component is down. The state of the system is determined by the state of its components through the structure function: $\varphi(X)$, where $X = X_1, \ldots, X_n$, and $\varphi(X) = 1$ if the system is up and $\varphi(X) = 0$ if the system is down.

> *Reliability* of a system, whose components are independent and non-renewable[1], is defined:
>
> Let X_i be the RV that represents the state of component i, $\varphi(X) = \varphi(X_1, \ldots X_n)$ is also a binary RV and the system reliability R is the probability that $\varphi(X) = 1$.

We make the analogy with these concepts in the following way:

- our system is the DIS
- our components are the sources
- each component is operational when it satisfies the accepted configurations
- the system is operational when its quality requirements are being satisfied
- *DIS Quality Certainty* is the *Reliability* of our system

[1] If a component fails it is not repaired or substituted

In order to calculate DIS Quality Certainty, in the following we deduce how our structural function φ is.

In previous Section we defined the RV Y_i that represents the accepted configurations satisfaction by a source S_i. Considering n sources, for a restriction vector $\nu = <r_1,\ldots,r_n>$ we have a set of n RV Y_i, i=1..n, each of which is equal to 1 if source S_i is operational (if the quality value at source i satisfies the restriction r_i), and is equal to 0 if S_i is not operational. Note that Y_i are statistically independent random variables (each source quality value varies independently from each other).

According to [Gertsbakh-89] a series system is a system that is operational if and only if all of its elements are up, while a parallel system is a system that is down if and only if all of its elements are not operational. Figure 4.5 illustrates these ideas. In a series system the structural function and its probability of being equal to 1 are the following:

$$\varphi(X) = \prod_{i=1..n} X_i$$

$$P(\varphi(X) = 1) = \prod_{i=1..n} P(X_i = 1)$$

In a parallel system they are:

$$\varphi(X) = 1 - \prod_{i=1..n} (1 - X_i)$$

$$P(\varphi(X) = 1) = 1 - \prod_{i=1..n} (1 - P(X_i = 1))$$

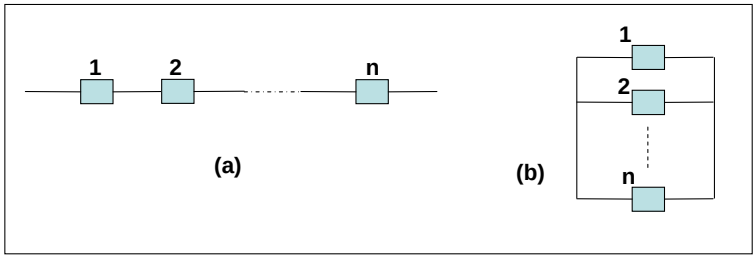


Figure 4.5: (a) Series system, (b) Parallel system

We must make an important distinction between two cases:

In the cases where the accepted configurations consist of **only one restriction vector**, we have a series system, and our structural function is $\varphi(Y) = \prod_{i=1..n} Y_i$. Therefore we calculate the DIS Quality Certainty as follows:

$$C = P(\varphi(Y) = 1) = \Pi_{i=1..n} P(Y_i = 1)$$

Once we have the Certainty calculated, we compare it to the probabilities expressed in the "probability" requirements. All these requirements are considered in the accepted configurations and their associated probabilities should be less or equal to the Certainty of the DIS.

Example 4.7

Retaking the Example 4.4, we now consider that the user posed the requirements accompanied by a probability value. See Figure 4.6.

Using an informal notation, the restriction vector was: < 5.5, 20, 5.5, 19.5, 5.5 >. We discretize the values as explained in previous Section: < 5, 20, 5, 19, 5 >

Suppose that from the sources models we calculate the following satisfaction probabilities:

$P(Y_1 = 1) = P(X_1 \leq 5) = 0.9$
$P(Y_2 = 1) = P(X_2 \leq 20) = 1$
$P(Y_3 = 1) = P(X_3 \leq 5) = 0.9$
$P(Y_4 = 1) = P(X_4 \leq 19) = 1$
$P(Y_5 = 1) = P(X_5 \leq 5) = 1$

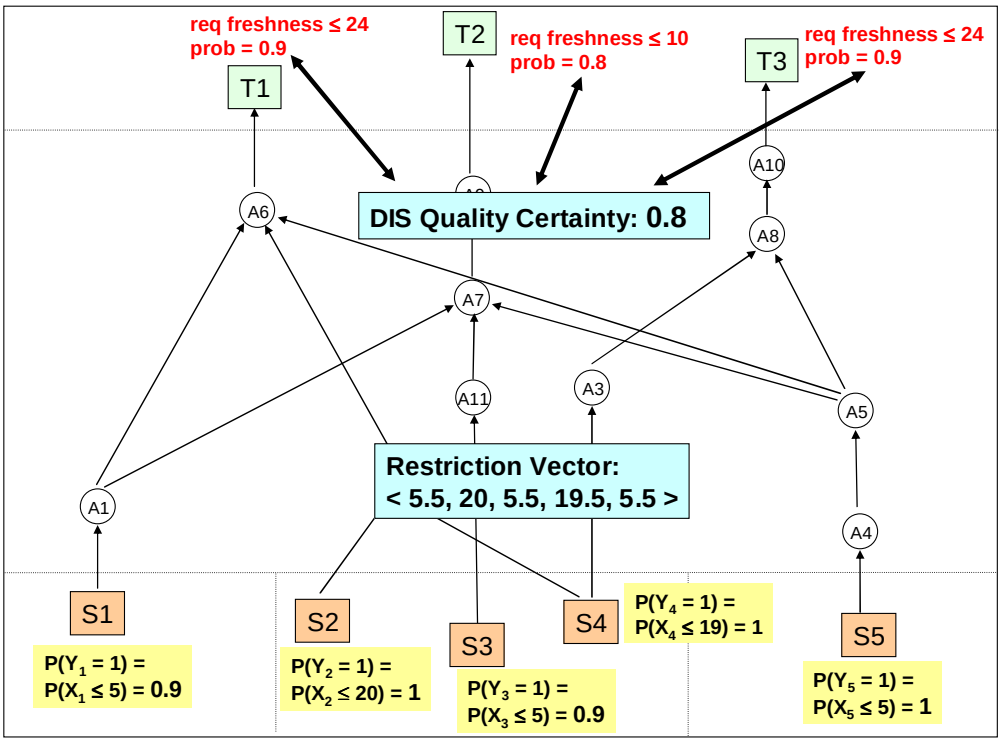


Figure 4.6: Calculation of DIS Quality Certainty

We calculate DIS Quality Certainty:

$C = P(\varphi(Y) = 1) = \Pi_{i=1..n} P(Y_i = 1) = P(Y_1 = 1)\, P(Y_2 = 1)\, P(Y_3 = 1)\, P(Y_4 = 1)\, P(Y_5 = 1)$

$C = 0.9 * 1 * 0.9 * 1 * 1 =$ **0.81**

0.8 is the probability that the DIS satisfies all the freshness values required. It satisfies T2 requirement, but it does not satisfy T1 and T3 probability requirements. However, we can calculate the probability that the DIS satisfies T1 requirement, this is the DIS Certainty taking into account only T1 requirement (C_{T1}). We do this in the following:

Restriction vector only for T1 requirement, sources S_1, S_2, S_4, S_5: < 20, 20, 20, 20 >

Now the satisfaction probabilities are:

$P(Y_1 = 1) = P(X_1 \leq 20) = 1$
$P(Y_2 = 1) = P(X_2 \leq 20) = 1$
$P(Y_4 = 1) = P(X_4 \leq 20 = 1$
$P(Y_5 = 1) = P(X_5 \leq 20) = 1$

Now C_{T1} **= 1**, which satisfies T1 requirement.

◊

In the case where the accepted configurations consist of **many restriction vectors**, we can think of a parallel system, where each vector is a component that may be operational or not. The problem we encounter is that the components **are not statistically independent**, since the probability of a restriction vector conditioned to the occurrence of other one, may be different from its marginal probability. We show this through an example:

Example 4.8

Consider the restriction vector for freshness: v1=<5, 10, 8> that pose the restrictions: $fr(S_1) \leq 5$, $fr(S_2) \leq 10$, and $fr(S_3) \leq 8$. Consider v2=<3, 9, 8>. We calculate the marginal probability P(v1) as:

$P(v1) = P(fr(S_1) \leq 5)\ P(fr(S_2) \leq 10)\ P(fr(S_3) \leq 8)$

On the other hand, we can state that:

P(v1|v2) = 1, since we know that if v2 occurs then v1 also occurs.

We cannot say that P(v1|v2) = P(v1), therefore v1 and v2 are not statistically independent.

◊

Due to the exposed reason, in the case of many restriction vectors that satisfy the requirements, we cannot calculate DIS quality certainty applying the technique proposed in [Gertsbakh-89] for parallel systems. We apply a different approach for these cases. We model the random experiment for this scenario in the following way.

Previous definition:

Definition 4.8: A *quality-values vector* is a set of quality values, containing one value for each one of the DIS sources. We use the following notation:

$vv = < qv_{S1}, \ldots, qv_{Sn} >$, where qv_{Si} is the quality value associated to source S_i □

Note that a source-restriction vector represents a set of quality-values vectors.

Random experiment:

- **Sample space.** It is the set of quality-values vectors that are possible in the DIS.
- **Event.** It is any subset of the sample space. A source-restriction vector is an event, since it represents a set of quality-values vectors.

We must calculate the probability that the accepted configurations are satisfied. This is the probability that at least one of the restriction vectors is satisfied.

Let $S = \{v_1, \ldots, v_m\}$, where v_i is a source restriction vector, be the accepted configurations.

We consider v_i as events of our experiment, being not disjoint. The probability of event v_1, or event v_2,…,or event v_m is the probability of the union of the events [Canavos-88].

To calculate the union of the events, considering that they are not-disjoint sets, in order to avoid multiple counting, we must apply the Inclusion-Exclusion Principle, from set theory, [Weisstein-03]:

Let |A| denote the cardinality of set A, then it follows immediately that

$|A \cup B| = |A| + |B| - |A \cap B|$

This formula can be generalized for sets A_1, …, A_p in the following manner.

$$|A_1 \cup A_2 \cup \ldots \cup A_p| = \sum_{1\le i\le p} |A_i| - \sum_{1\le i<j\le p} |A_i \cap A_j| + \sum_{1\le i<j<k\le p} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{p-1} |A_1 \cap A_2 \cap \ldots \cap A_p|$$

We apply the principle to Probability:

If there are only 2 events, m = 2, the formula is:

$$P(\nu_1 \cup \nu_2) = P(\nu_1) + P(\nu_2) - P(\nu_1 \cap \nu_2)$$

If there are more than 2 events:

$$P(\nu_1 \cup \nu_2 \cup \ldots \cup \nu_m) = \sum_{1\le i\le m} P(\nu_i) - \sum_{1\le i<j\le m} P(\nu_i \cap \nu_j) + \sum_{1\le i<j<k\le m} P(\nu_i \cap \nu_j \cap \nu_k) - \ldots + (-1)^{m-1} P(\nu_1 \cap \nu_2 \cap \ldots \cap \nu_m)$$

To calculate the probability of one restriction vector $P(\nu_i)$ we directly apply the calculation of the DIS Quality Certainty for only one restriction vector (presented above).

In order to calculate the probabilities of the intersections of restriction vectors, we previously solve the intersection operations. In the following we show that the result of these intersections is always a restriction vector.

Theorem 4.1

Given the accepted configurations of a DIS, the intersection between two restriction vectors is a restriction vector.

Demonstration:

We demonstrate by construction.

Let ν_1 and ν_2 be restriction vectors, such that:

$\nu_i = <r_{iS1}, \ldots, r_{iSn}>$, where r_{iSj} is the restriction associated to source S_j,
$r_{iSj} = <qfactor, S_j, op, value_i>$ (recall that *qfactor* and *op* are the same for all the restrictions of the same *accepted configurations*)

When *op* = "<" or *op* = "≤":

$\nu_3 = \nu_1 \cap \nu_2$: FOR j:1..n DO
IF $value_1 \le value_2$ THEN
$r_{3Sj} = <qfactor, S_j, op, value_1>$
ELSE
$r_{3Sj} = <qfactor, S_j, op, value_2>$

When *op* = ">" or *op* = "≥":

$\nu_3 = \nu_1 \cap \nu_2$: FOR j:1..n DO
IF $value_1 \ge value_2$ THEN
$r_{3Sj} = <qfactor, S_j, op, value_1>$
ELSE
$r_{3Sj} = <qfactor, S_j, op, value_2>$

The diagrams in Figure 4.7 show how the points that belong to the intersection are the ones that we obtain through the previous construction algorithm. We consider the possible kinds

of cases for two restriction vectors r1= <x1, y1> and r2= <x2, y2>. Figure 4.7, a) and b) are the cases where *op* = "≤". In a), x1< x2 and y1 < y2. In b), x1< x2 and y2 < y1. c) and d) are the cases where *op* = "≥". In c), x1< x2 and y1 < y2. In d), x1< x2 and y2 < y1.

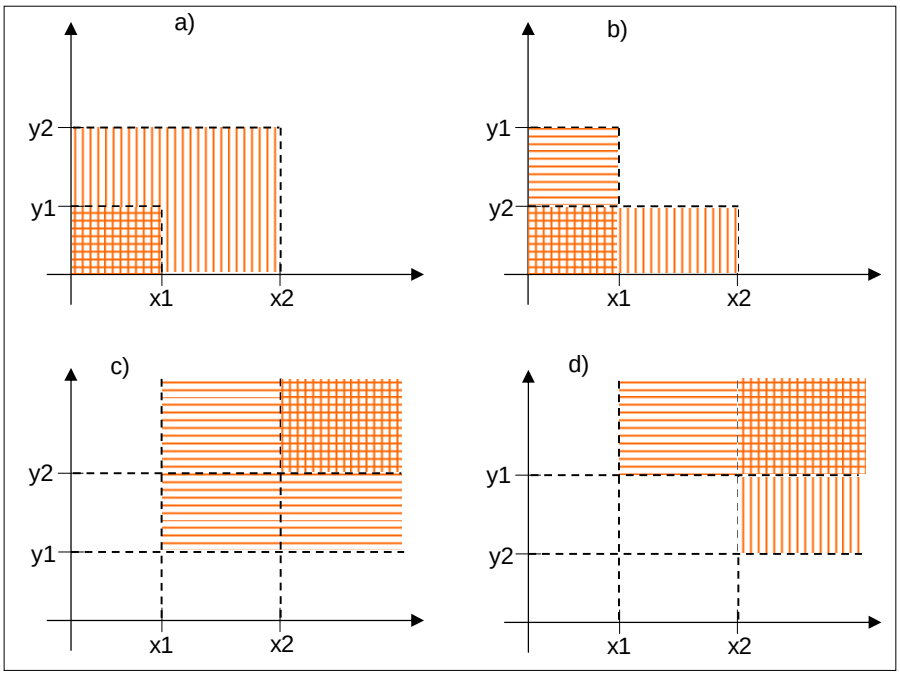


Figure 4.7: Intersection of restriction vectors <x1,y1> and <x2,y2>. a), b): restriction vectors with ≤ operator. c), d): restriction vectors with ≥ operator.

∇

As the result of the intersections is a restriction vector, we first solve these operations and then we solve the probability of the resulting restriction vector.

In summary, in the cases where the accepted configurations consist of **many restriction vectors,** $\nu_1, \ldots, \nu_m$, we calculate the DIS Quality Certainty as follows:

If m = 2:

$$C = P(\nu_1) + P(\nu_2) - P(\nu_1 \cap \nu_2)$$

If m >2:

$$C = \sum_{1 \le i \le m} P(\nu_i) - \sum_{1 \le i < j \le m} P(\nu_i \cap \nu_j) + \sum_{1 \le i < j < k \le m} P(\nu_i \cap \nu_j \cap \nu_k) - \ldots + (-1)^{m-1} P(\nu_1 \cap \nu_2 \cap \ldots \cap \nu_m)$$

Example 4.9

Consider a DIS where there are 5 sources S1… S5. We want to calculate the Certainty for the quality factor accuracy. The Accepted Configurations consist of the following restriction vectors, whose operator is ≥:

v1 = <0.8, 1, 0.8, 0.9, 1> (This means the restrictions: accuracy(S1)≥0.8, accuracy(S2)≥1, accuracy(S3)≥0.8, accuracy(S4)≥0.9, accuracy(S1)≥1)

v2 = <1, 1, 0.8, 0.7, 0.9>

v3 = <0.9, 1, 0.8, 0.8, 1>

v4 = <0.9, 0.9, 0.8, 1, 0.9>

$C = \sum_{1\le i\le m} P(v_i) - \sum_{1\le i<j\le m} P(v_i \cap v_j) + \sum_{1\le i<j<k\le m} P(v_i \cap v_j \cap v_k) - \ldots + (-1)^{m-1} P(v_1 \cap v_2 \cap \ldots \cap v_m)$

C = P(v1) + P(v2) + P(v3) + P(v4) – (P(v1 ∩ v2) + P(v2 ∩ v3) + P(v3 ∩ v4) + P(v1 ∩ v3) + P(v1 ∩ v4) + P(v2 ∩ v4)) + P(v1 ∩ v2 ∩ v3) + P(v2 ∩ v3 ∩ v4) + P(v1 ∩ v3 ∩ v4) – P(v1 ∩ v2 ∩ v3 ∩ v4)

We have the probability distribution of each source, where the RV is X_i for source Si.

P(v1) = P($X_1 \ge 0.8$) P($X_2 \ge 1$) P($X_3 \ge 0.8$) P($X_4 \ge 0.9$) P($X_5 \ge 1$)

P(v2) = P($X_1 \ge 1$) P($X_2 \ge 1$) P($X_3 \ge 0.8$) P($X_4 \ge 0.7$) P($X_5 \ge 0.9$)

P(v3) = P($X_1 \ge 0.9$) P($X_2 \ge 1$) P($X_3 \ge 0.8$) P($X_4 \ge 0.8$) P($X_5 \ge 1$)

P(v4) = P($X_1 \ge 0.8$) P($X_2 \ge 0.9$) P($X_3 \ge 0.8$) P($X_4 \ge 1$) P($X_5 \ge 0.9$)

v1∩v2 = <1, 1, 0.8, 0.9, 1>

P(v1∩v2) = P($X_1 \ge 1$) P($X_2 \ge 1$) P($X_3 \ge 0.8$) P($X_4 \ge 0.9$) P($X_5 \ge 1$)

v2∩v3 = <1, 1, 0.8, 0.8, 1>

P(v2∩v3) = P($X_1 \ge 1$) P($X_2 \ge 1$) P($X_3 \ge 0.8$) P($X_4 \ge 0.8$) P($X_5 \ge 1$)

(v3∩v4) = <0.9, 1, 0.8, 1, 1>

P(v3∩v4) = P($X_1 \ge 0.9$) P($X_2 \ge 1$) P($X_3 \ge 0.8$) P($X_4 \ge 1$) P($X_5 \ge 1$)

We calculate analogously the probabilities of the other intersections. From the probability distributions of each source we substitute the corresponding probability values, obtaining C.

◊

4.3 Satisfaction of non-probabilistic user quality requirements

When there are quality requirements whose type are “maximum”, “minimum”, “average” or “frequency”, we propagate them to the sources, obtaining the accepted configurations. From the sources models we obtain the corresponding calculated values, e.g expectation, maximum, etc, and verify if they satisfy the accepted configurations. It is important to remember that the requirement

type “average” corresponds to the concept of expectation in source models, and the requirement type “frequency” corresponds to the concept of mode in source models.

In the following we sketch the algorithm used for determining the satisfaction of non-probabilistic requirements. We must execute it for each type of the existing quality requirements, and previously calculate the corresponding accepted configurations. Given a requirement type and the accepted configurations for the requirements of this type, the algorithm determines if the sources satisfy the accepted configurations. For doing this, it verifies if exists a vector in the accepted configurations whose source restrictions are all satisfied.

```
FUNCTION QualitySatisfaction (G: QualityGraph, ac: AcceptedConfigurations, qfactor:
String, reqt: RequirementType): Boolean

sat1, sat2: Boolean

sat2 = FALSE
FOR EACH restriction vector v of ac.rv-space DO
        sat1 = TRUE
        FOR EACH restriction r_Si of v DO
                sat1 = Satisfy? ( qfactor, reqt, r_Si) AND sat1
        IF sat1 THEN sat2 = TRUE
QualitySatisfaction = sat2

END FUNCTION
```

```
FUNCTION Satisfy? (qfactor: String, reqt: RequirementType, r: SourceRestriction):
Boolean

val: Decimal
CASE reqt OF
        “maximum”:      val = ObtainMaximum (G, qfactor, GetSource(r))
                        CASE GetOp(r) OF
                                “<”: Satisfy? = ( val < GetValue(r) )
                                “≤”: Satisfy? = ( val ≤ GetValue(r) )
                                “=”: Satisfy? = ( val = GetValue(r) )
        “minimum”:      val = ObtainMinimum (G, qfactor, GetSource(r))
                        CASE GetOp(r) OF
                                “>”: Satisfy? = ( val > GetValue(r) )
                                “≥”: Satisfy? = ( val ≤ GetValue(r) )
                                “=”: Satisfy? = ( val = GetValue(r) )
        “average”:      val = ObtainExpectation (G, qfactor, GetSource(r))
                        CASE GetOp(r) OF
                                “<”: Satisfy? = ( val < GetValue(r) )
                                “≤”: Satisfy? = ( val ≤ GetValue(r) )
                                “>”: Satisfy? = ( val > GetValue(r) )
                                “≥”: Satisfy? = ( val ≤ GetValue(r) )
```

```
                    "=": Satisfy? = ( val = GetValue(r) )
"frequency":    val = ObtainMode (G, qfactor, GetSource(r))
                CASE GetOp(r) OF
                    "<": Satisfy? = ( val < GetValue(r) )
                    "≤": Satisfy? = ( val ≤ GetValue(r) )
                    ">": Satisfy? = ( val > GetValue(r) )
                    "≥": Satisfy? = ( val ≤ GetValue(r) )
                    "=": Satisfy? = ( val = GetValue(r) )
END FUNCTION
```

The functions **ObtainMaximum**, **ObtainMinimum**, **ObtainExpectation** and **ObtainMode**, calculate the maximum, minimum, expectation and mode, respectively from the source model, as explained in Section 3.

The functions **GetSource, GetOp and GetValue** obtains the source, the operator and the value, respectively, from the source restriction.

As can be seen in the presented pseudo-code, to verify the satisfaction of a source restriction (function **Satisfy?**), we compare the value calculated from the source model (maximum, minimum, expectation or mode) to the value of the source restriction.

4.4 Probability Distribution of DIS Quality

In addition to calculating the quality provided by the DIS for certain user quality requirements (as presented in previous Sections), we can calculate the probabilities of the different quality values that the DIS may give. For simplicity we consider this calculation for only one data target, and therefore only the sources involved in its generation.

We calculate the DIS quality values that result from all the possible combinations of sources quality values, and the probabilities of satisfying them. This can be done if and only if the set of possible quality values in each source is finite.

For the calculation of the probability of each DIS quality value we calculate the probability that one of the combinations is satisfied by the sources. For this, we sum the probabilities of the combinations of sources values (quality-values vectors) that give the DIS quality value, since we consider the quality-values vectors as disjoint events.

The following are the steps to claculate the probability distribution of DIS quality, for target T and set of sources (involved in T generation) $\{S_1, \ldots, S_n\}$:

1- Generate all the possible combinations of quality values for $S_1, \ldots, S_n$, obtaining a set of combinations *Comb* = $\{vv_1, \ldots, vv_k\}$, where vv_i is a quality-values vector (defined in Section 4.2.2) $vv_i = \{q_{i1}, \ldots, q_{in}\}$

2- For each $vv_i \in$ *Comb*, calculate the quality value provided in T, qv_i, generating *DISValues1* = $\{qv_1, \ldots, qv_k\}$. This is done through the corresponding quality evaluation algorithm (Chapter 3).

3- Eliminate duplicate values from *DISValues1*, generating *DISValues2* = $\{qv_{i1}, \ldots, qv_{im}\}$, $1 \leq ij \leq k$.

4- For each $qv_{ij} \in$ *DISValues2*, sum the probabilities of the vectors of *Comb* that generate qv_{ij} (this probabilities are obtained from the sources' models), obtaining the probability that one of the vectors is satisfied by the sources.

The following example illustrates how the steps are applied.

Example 4.10

Continuing with example 4.4, where T3 obtains its data from sources S4 and S5, we will calculate the possible DIS maximum freshness values, in this case T3 freshness values, and their corresponding probabilities. See Figure 4.7.

Suppose that:

S4 possible freshness values: {0, 1, 2, 3, 4, 5, 6}

S5 possible freshness values: {0, 1, 2}

The following is the set of all possible combinations of freshness values for S4, S5:

Comb ={ <0, 0>; <0, 1>; <0, 2>; <1, 0>; <1, 1>; <1, 2>; <2, 0>; <2, 1>; <2, 2>; <3, 0>; <3, 1>; <3, 2>; <4, 0>; <4, 1>; <4, 2>; <5, 0>; <5, 1>; <5, 2>; <6, 0>; <6, 1>; <6, 2> }

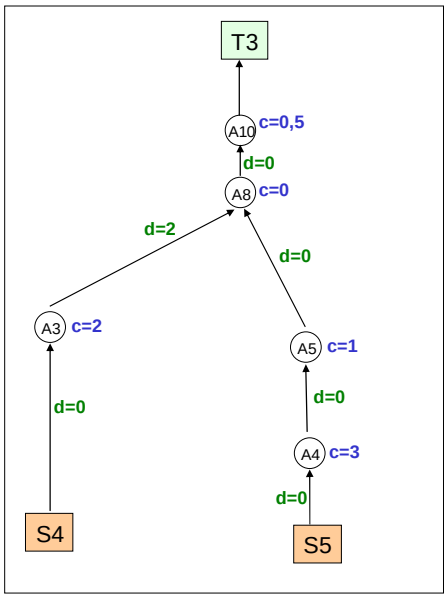


Figure 4.7: Quality Graph for T3

For each combination, we calculate the quality value provided in T3. As stated in Example 4.4, the freshness propagation function for a node A was:

Freshness(A) = Max (Freshness(A_1) + Delay(A_1,A), …, Freshness(A_n) + Delay(A_n,A)) + Cost(A), where A_1, …, A_n are A predecessors

For simplicity of the presentation we show only some of the combinations:

Sources freshness: <S4, S5>	T3 Freshness:
<0, 0>	4.5
<0, 1>	5.5
<0, 2>	6.5
<1, 0>	5.5

<1, 1>	5.5
……	……
<6, 1>	10.5
<6, 2>	10.5

We eliminate duplicated values of T3 freshness:

{4.5, 5.5, 6.5, …, 10.5}

For fr(T3) = 4.5:

P(fr(T3) = 4.5) = P(<0, 0>) = P(fr(S4) = 0) P(fr(S5) = 0)

For fr(T3) = 5.5:

P(fr(T3) = 5.5) = P(<0, 1>) + P(<1, 0> + P(<1, 1>) =

P(fr(S4) = 0) P(fr(S5) = 1) + P(fr(S4) = 1) P(fr(S5) = 0) + P(fr(S4) = 1) P(fr(S5) = 1)

The models of S4 and S5 include this information:

P(fr(S4) = 0) = 0.7

P(fr(S4) = 1) = 0.2

P(fr(S5) = 0) = 0.6

P(fr(S5) = 1) = 0.4

P(fr(T3) = 4.5) = 0.7 * 0.6

P(fr(T3) = 5.5) = 0.7 * 0.4 + 0.2 * 0.6 + 0.2 * 0.4

Distribution of T3 freshness values:

T3 Freshness	Probability
4.5	0.42
5.5	0.48
…….	…….

◊

From the probability distribution of the possible DIS quality values we can easily obtain the DIS Quality Certainty of a possible DIS quality value *v1* (considering *v1* as the user quality requirement), i.e the probability that the DIS quality value is equal or "better" than *v1*. Depending on the quality factor, we consider *v2* as better than *v1* if $v2 \leq v1$ or if $v2 \geq v1$. In particular, if the quality factor is freshness it should hold $v2 \leq v1$, and if it is accuracy it should hold $v2 \geq v1$.

Let X be a RV representing the possible DIS quality values.

Let the probability distribution of RV X be the following:

$P(X=qv_1) = p_1$

$P(X=qv_2) = p_2$

…..

$P(X=qv_n) = p_n$

We calculate the DIS Quality Certainty of qv_i as:

$P(X \leq qv_i) = \sum_{j:1..i} p_j$

or

$P(X \geq qv_i) = \sum_{j:i..n} p_j$

according to the quality factor.

4.5 Models Specification

We specify two types of DIS quality behavior models: (i) DIS quality satisfaction model and (ii) DIS quality distribution model. (i) includes DIS Certainty and satisfaction of non-probabilistic quality requirements. (ii) is for specifying probability distribution models of the quality provided by the DIS in certain data-target.

Definition 4.9: A *DIS quality satisfaction model* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, requirements-set, sat-probability>, where

- qfactor is a String, representing the quality factor,
- requirements-set is a set of Requirements,
- sat-probability is a number between 0 and 1, representing the probability of the satisfaction of these requirements by the DIS. □

We include DIS quality satisfaction models in the *quality graph* (presented in Chapter 3), as *properties* associated to the graph.

In the case of models of satisfaction of non-probabilistic quality requirements, the sat-probability field is always 1.

Definition 4.10: A *DIS quality distribution model* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, data-target, distribution>, where

- qfactor is a String, representing the quality factor,
- data-target is a String, the name of the data target to which the model corresponds,
- distribution is a set of pairs <qvalue, probability>, where
    - qvalue is a Decimal number and
    - probability is a number between 0 and 1, representing the probability of the quality value.

□

We include DIS quality distribution models in the *quality graph*, each one as a *property* associated to a data-target.

5. Quality Behaviour through Time

The behaviour of the sources quality factors, i.e. the way their values vary, may change through time, and therefore their probabilistic models. For instance, in the case of freshness, the update period of a periodically updated source may change, or if the updates follow a Poisson distribution, it may change the update frequency λ. In the case of any quality factor that is measured periodically and its distribution is estimated, this distribution may change when considering new measurements.

We propose to maintain for each source, the history of the different distributions, a quality factor has through time. We simply store some summary values that characterize the probability distribution, such as the expectation, the mode, the maximum and minimum values, with a corresponding timestamp.

Maintaining this information is very useful because it constitutes highly valuable information when we have to take actions in order to improve the DIS quality.

The following is the specification of this historical information.

Definition 4.11: A *source quality models history* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, source, distributions>, where

- qfactor is a String, representing the quality factor,
- source is a String, the name of the source to which the model corresponds,
- distributions is a set of 5-uples, indicators = <minimum, maximum, expectation, mode, timestamp>, where
    - minimum, maximum, expectation and mode are Decimal numbers,
    - timestamp is a date/time field, representing the instant when the distribution was stated as current. □

The indicators are the information that characterizes a distribution.

The most recent distribution in a *source quality models history* corresponds to the current distribution of the source, that is, to the *source quality behavior model.*

We include *source quality models histories* in the *quality graph* (presented in Chapter 3), each one as a *property* associated to a source.

6. Summary

The way we have chosen to maintain information about the quality of the DIS is through quality behavior models. These models show how quality behaves beyond punctual values, providing information that enables to make estimations about the current and future quality.

The quality behavior models are probabilistic models. We apply them to sources quality as well as to DIS quality. We characterize DIS quality, through two different approaches, as: the quality values that the DIS can take (for the data targets), and the quality considering the user quality requirements, i.e. the satisfaction or not of the quality requirements.

For DIS quality approach where user quality requirements are taken into account, it is necessary to define and be able to calculate the accepted configurations. These are the quality requirements at the sources that are deduced from the user requirements and the transformations suffered by the data as it goes from sources to data targets. The complexity of these requirements is due to the possible existence of many combinations of source quality values that satisfy the user quality requirements. This caused that, when modeling DIS quality, we first stated a solution for the case where there is

only one restriction for each source (only one restriction vector), and then we achieved a more advanced solution for the cases where the accepted configurations consist of many restrictions vectors.

For the DIS quality model that provides the probability distribution for the possible DIS quality values, it also was considered that it may exist many combinations of sources quality values that generate the same quality value in a data target, obtaining a global probability in those cases.

The source quality models that were presented are examples of possible models. In some cases it may happen that none of these models are useful or adequate, and it may exist other ones, which better adapt to the cases. The idea is to show the followed approach and the possibility of working with this kind of models.

All the presented models can be summarized into values that characterize them, such as the expectation or the mode. We propose to store the summarized information of the models, through time, since they may be used to explain current situations and observing tendencies.

The examples presented all along the chapter are intended to clarify the proposals and show their viability and usefulness.

CHAPTER 5. QUALITY CHANGES DETECTION

We must be alert. Our Data Integration System may loose quality…

1. Introduction

Detecting changes implies identifying the changes that are relevant to our problem. Our problem can be synthesized as achieving the satisfaction of the DIS user quality requirements as continuously as possible. Therefore, we are interested in detecting changes that generate the dissatisfaction of DIS quality requirements.

There are different kinds of changes that may affect DIS Quality. We manage changes on sources quality, changes on DIS structure, and changes on quality requirements. We present a classification for changes, according to their nature and to the components of the system where they occur. On the other hand, changes may have different degrees of incidence on the DIS, characteristic that is very important for detecting relevant changes. Therefore, we define another classification, orthogonal to the first one, which considers how the change affects DIS quality.

In order to model the dynamism involved in the occurrence of changes and its notification to the different modules of our management system, we base ourselves on the notion of *events*. We define different types of events. We consider that some of them are generated by the sources, while there are many of them that are generated internally to the Quality Management System (as said before, QMS).

With respect to the events received from the sources, our intention is to leave this aspect as open as possible, since the quantity of meta-information a source may give to a DIS is very variable. It depends on the degree of autonomy of the source as well as its will of collaborating with the DIS and sharing the different information. In our proposal, we consider some possible cases of meta-information given by sources.

Our proposal for relevant changes detection is mainly contained in the management of events that are generated internally to the QMS. We manage these events in a way that changes are gradually filtered, remaining only the relevant ones. For the management of events we propose a set of ECA (Event Condition Action) rules, which we call *Change Detection Rules*.

There is a close relationship between the quality models developed in the previous chapter and the techniques for detecting relevant quality changes proposed in the present chapter. First, the source quality change that is detected and then treated by the rules is a change on the *quality model* of the source. Second, the evaluation of the relevance of any change, performed through the rules, is strongly based on the *quality models* of the DIS (verifying DIS Quality Certainty, and satisfaction of requirements of average, most frequent value, etc.).

The detection of relevant quality changes involves the verification of different aspects of quality satisfaction. Our proposal is to verify only the aspects that were affected by the change, in order to avoid unnecessary work. Therefore, when the management system is normally working, partial quality verifications are done each time a change occurs. However, it is necessary to define how a complete verification of DIS quality must be done, stating this way, the quality level we demand to the DIS and the state we try to maintain in our proposal. On the other hand, this complete quality verification may be done periodically to ensure the detection of quality problems that may be involuntary ignored through the change detection process. We propose one possible complete verification for DIS quality.

Figure 5.1 shows the processes that are proposed for solving changes detection, and their interaction. Events that come from sources are processed, sometimes causing updates to the source quality models contained in the *estimations & statistics* meta-information. Certain changes in this meta-information generate the creation of QMS events. These events may also be created as a consequence of an execution of a complete DIS quality verification. QMS events are processed by the Change Detection Rules, which sometimes create new QMS events, and other times generate events that notify to the Quality Repair module of the QMS, that a relevant quality change occurred.

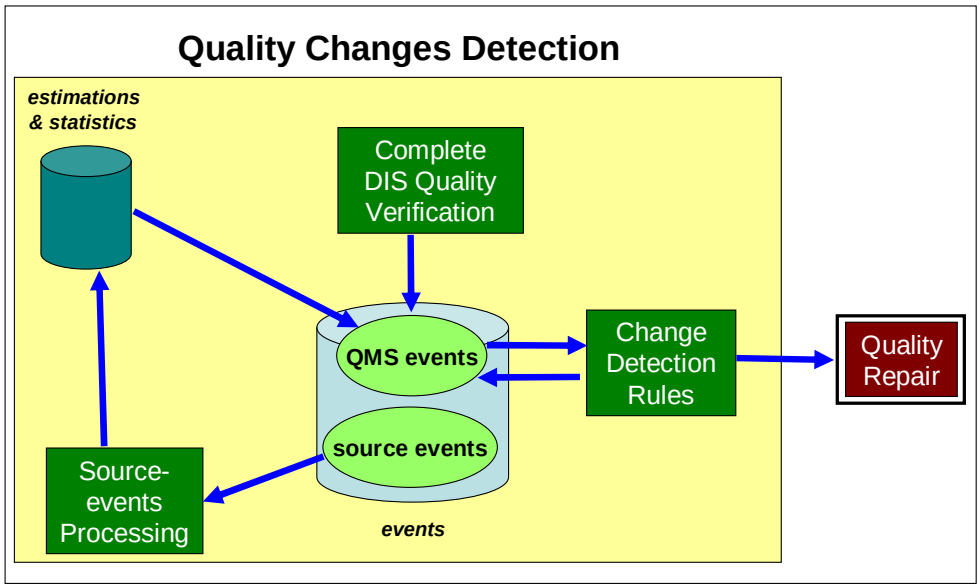


Figure 5.1: Quality Changes Detection Architecture

This Chapter is organized as follows: Section 2 presents a proposal for doing a complete DIS quality verification, Section 3 presents a taxonomy of changes, Section 4 presents the events defined for change management, Section 5 presents a method for relevant changes identification, Section 6 present an example and Section 7 presents the summary of the chapter.

2. Complete DIS-Quality Verification

Before addressing the detection of quality changes, we define how a complete quality verification must be done for a quality factor. Later, we take this as a guide for the different verifications that must be done in the different cases of quality changes.

The algorithm for a complete DIS-quality verification must perform the following verifications:

- The accepted configurations that have as requirements all DIS user quality requirements of type “minimum” must be satisfied
- The accepted configurations that have as requirements all DIS user quality requirements of type “maximum” must be satisfied
- The accepted configurations that have as requirements all DIS user quality requirements of type “average” must be satisfied

- The accepted configurations that have as requirements all DIS user quality requirements of type “frequency” must be satisfied
- DIS Quality Certainty must be greater or equal to the probabilities of all DIS user quality requirements of type “probability”

We present a pseudo-code of the algorithm, for a given quality graph and a given quality factor:

```
FUNCTION DIS_Quality_Verification (G: QualityGraph, qfactor: String): Boolean

DIS_Quality_Verification =      Quality_Verification (G, qfactor, “minimum”,
                               “ac_minimum”) AND Quality_Verification (G, qfactor,
                               “maximum”, “ac_maximum”) AND Quality_Verification (G,
                               qfactor, “average”, “ac_expectation”) AND  Quality_Verification
                               (G, qfactor, “frequency”, “ac_mode”) AND
                               Quality_Certainty_Verification (G, qfactor)
END FUNCTION
```

```
FUNCTION Quality_Verification (G: QualityGraph, qfactor: String, reqtype: String,
                                   acname: String): Boolean

reqs: Set of <Target, String>
ac: AcceptedConfigurations

reqs = Get_Target_Requirements (G, qfactor, reqtype)
IF NOT Empty (reqs)
      ac = Accepted_Conf_Calculation (G, qfactor, reqs)
      G = Add_Graph_Property (G, “AcceptedConfigurations”, acname, ac)
      Quality_Verification = QualitySatisfaction (G, ac, qfactor, reqtype)
ELSE
      Quality_Verification = TRUE

END FUNCTION
```

```
FUNCTION Quality_Certainty_Verification (G: QualityGraph, qfactor: String, acname:
                                             String): Boolean

reqs: Set of <Target, String>
ac: AcceptedConfigurations
aux: Boolean
C: Decimal (0 .. 1)

reqs = Get_Target_Requirements (G, qfactor, “probability”)
IF NOT Empty (reqs)
      ac = Accepted_Conf_Calculation (G, qfactor, reqs)
      G = Add_Graph_Property (G, “AcceptedConfigurations”, acname, ac)
      C = DIS_Quality_Certainty (G, qfactor, ac)
      aux = TRUE
      FOR EACH requirement r of reqs DO
```

```
            aux = aux AND ( C ≥ GetProbability (r) )
        Quality_Certainty_Verification = aux
    ELSE
        Quality_Certainty_Verification = TRUE
    END FUNCTION
```

Function **Get_Target_Requirements** obtains, given a quality graph and a quality factor, the names of all the requirements of certain type. It returns a set of pairs <target, requirement-name>.

Function **Accepted_Conf_Calculation** was specified in Chapter 4, Section 4.1. It calculates the accepted configurations for a quality graph, a set of requirements and a quality factor.

Function **Add_Graph_Property** receives a graph, a property type, a property name, and a property value, and modifies the graph adding the new property.

Function **QualitySatisfaction** was specified in Chapter 4, Section 4.3. Given a requirement type and the accepted configurations for a set of requirements of this type, this function determines if the sources satisfy the accepted configurations.

Function **DIS_Quality_Certainty** implements the formula stated in Chapter 4, Section 4.2.2 to calculate DIS Quality Certainty from the accepted configurations and the models of the involved sources.

Function **GetProbability** obtains the probability from a requirement of type "probability".

The QMS executes periodically this verification in the DIS, since there may be changes that are not detected through the proposed mechanism although they generate requirements dissatisfaction. If this execution detects dissatisfaction of quality requirements, the events *QValuesDissatisfaction* and/or *ProbabilityDissatisfaction*, which are explained later in this chapter, are generated.

3. Changes Taxonomy

We classify the changes that may affect the DIS quality, according to the taxonomy shown in Figure 5.2. In the following we describe each type of change:

- *Quality change*. A change on the value/s associated to a quality factor.
  - *Change on the quality of a source.* Change in quality behavior model.
  - *Change on user quality requirements.* Change on one user quality requirement, addition of a user quality requirement or elimination of a user quality requirement.
- *Structural change*. Change on some structure or property of the DIS.
  - *Source-schema change*. Changes on the structure of a source table, addition of a source table or elimination of a source table.
  - *Integrated-schema change*. For the cases where the DIS has a pre-defined integrated schema. Changes on a table structure, addition of a table or elimination of a table.
  - *Change on the DIS transformation graph*. Change on the structure of the graph or change on an activity's property (cost, effectiveness, etc.).

We left out of the scope of this work the management of source schema and integrated schema changes.

The possible changes of each type of change, are at the same time classified according to how they affect the DIS.

From all the possible source quality changes (See Figure 5.3, (a)), only a subset change the source quality model, since there are many changes that maintain the same probability distribution of the source quality. From the mentioned subset only some changes affect the DIS quality, changing the quality values of the information received by the user. From the changes that change the DIS quality, only some make the user quality requirements dissatisfied. The following summarizes this classification of changes.

- Source quality changes (Figure 5.3, (a)).
  - Changes that change the source quality model
    - Changes that affect the DIS quality
      - Changes that make the user quality requirements dissatisfied
      - Changes that do not make the user quality requirements dissatisfied
    - Changes that do not affect the DIS quality
  - Changes that do not change the source quality model

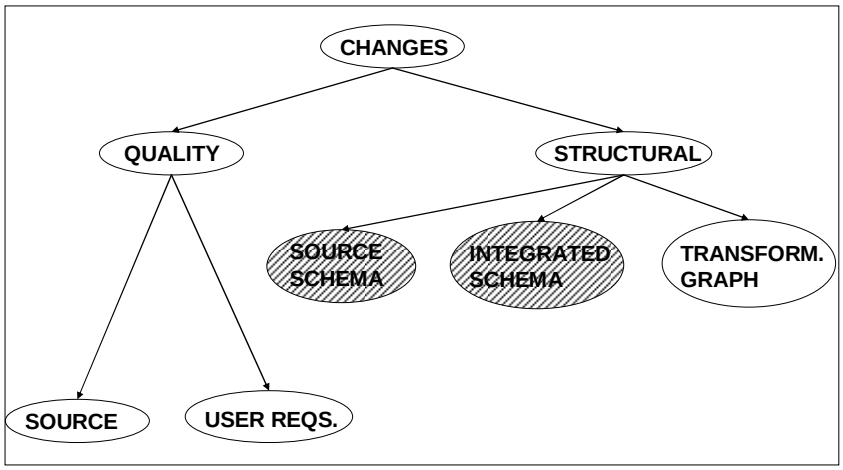


Figure 5.2: Changes taxonomy

From all the possible changes on the transformation graph (See Figure 5.3, (b)), only some change the DIS quality, and from these, only some generate the quality requirements dissatisfaction. For example, if a cleaning activity is eliminated, the accuracy of some query results may decrease, which may generate the dissatisfaction of the user quality requirements. In the same way, if this activity is added to the system the accuracy may increase, and this would change the DIS quality but not generate requirements dissatisfaction. The following summarizes this classification of changes.

- Transformation graph changes (Figure 5.3, (b)).
  - Changes that affect the DIS quality
    - Changes that make the user quality requirements dissatisfied
    - Changes that do not make the user quality requirements dissatisfied

- Changes that do not affect the DIS quality

Only some of all the possible changes on user quality requirements (See Figure 5.3, (c)) generate changes on the sources required quality values (i.e., on the accepted configurations). For example, if user quality requirement *req1* changes, there may exist another requirement *req2* that already generated a stronger requirement on the sources, so *req1* change does not affect sources required values. Besides, only some of the changes that affect sources required values cause the dissatisfaction of the user quality requirements, since it may happen that the sources quality values satisfy the new requirements. The following summarizes this classification of changes.

- User quality requirements changes (Figure 5.3, (c)).
  - Changes that affect the sources required quality values
    - Changes that make the user quality requirements dissatisfied
    - Changes that do not make the user quality requirements dissatisfied
  - Changes that do not affect the DIS quality

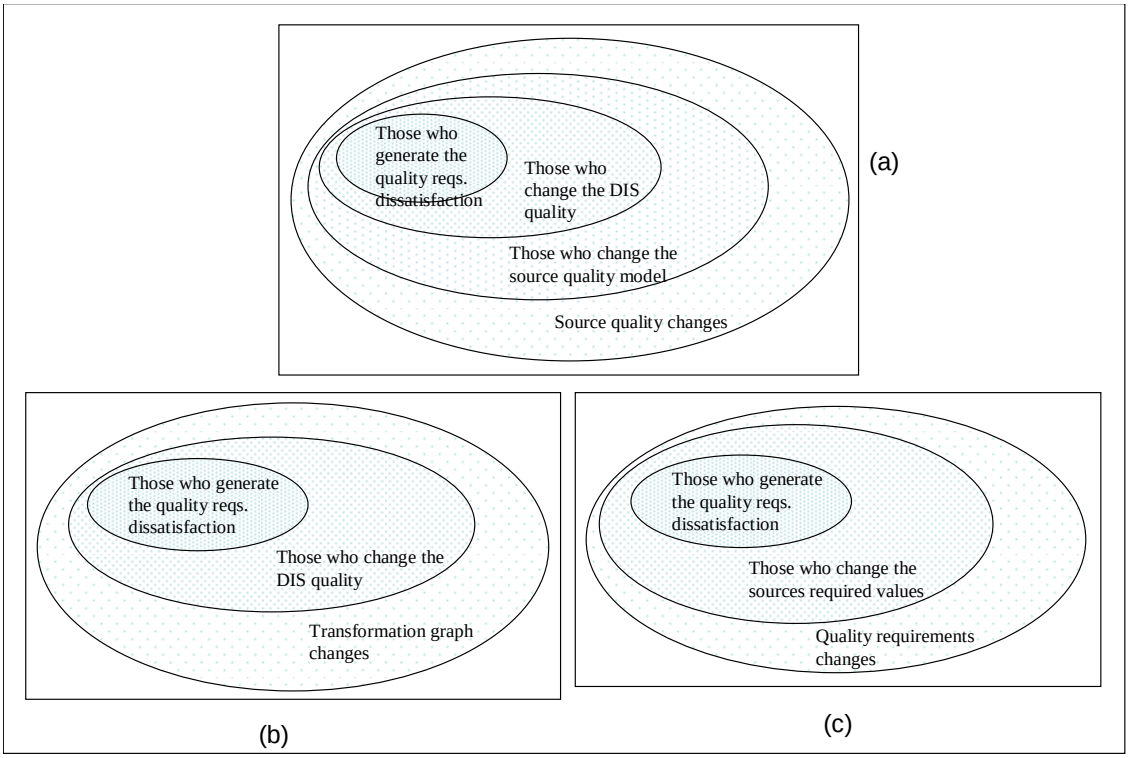


Figure 5.3: (a) Source changes classification, (b) Transformation graph changes classification, (c) User quality requirements changes classification.

Our strategy to detect relevant changes is based on this latest classification. We filter the changes through the different layers shown in Figure 5.3, obtaining the ones that generate quality requirements dissatisfaction.

4. Events

The way the QMS (as said before, Quality Management System) is notified about changes and also the way it treats these changes is through events. We lean our change detection mechanism on the reception, generation and management of events.

We model the events using the notions of classes and objects of the Object Oriented paradigm. We define different classes of events whose objects are the particular events that are generated in the system.

We define two groups of events: (1) the events that come from the sources to the DIS and (2) the events that are generated at the QMS. The events from (1) are, for example, source data updates and source schema changes. The events from (2) are changes of DIS elements, such as a source model or transformation graph change, which need to be noticed and managed.

Events from group (1) are generated at the sources and captured by the QMS, while group (2) events are generated by the QMS and also captured by it.

We concentrate in the management of group (2) events, since this is the base of our approach for relevant changes detection.

4.1 Events that come from sources

The events generated at the sources enable the notification to the QMS of sources changes that are important for it. We assume that these events are generated by the sources themselves, although they could be generated by an external monitoring mechanism.

These events may be generated as consequence of the following happenings:

- There was a data update
  There was a data update at the source. Note that a source for us is a whole data source or a relation from a relational source (a source relation).
- Update frequency λ has changed
  In the cases where the source is updated according to a Poisson process, the source may give the information about the estimated update frequency λ, notifying the QMS when λ changes.
- Update period has changed
  In the cases where the source is updated periodically, the source may inform the QMS when the period changes.

For freshness as well as for accuracy quality factors, the QMS needs to detect some of the above-mentioned sources' events, or some other one that provides useful information, in order to obtain a periodic measurement of the freshness or of the accuracy. For this work, we assume that the QMS is able to detect at least data update events at the sources.

In Table 5.1 we show the events classes that we define.

We do not extend in the details of these classes, neither in the management of these events, since we prefer to concentrate in the management of the QMS events. In the following we give a brief description of the processing of these events.

Event Name	**Event Description**
SourceDataUpdate	There was a data update in the source.
UpdateFreqChange	The estimated update frequency (λ) of a Poisson-updated source has changed.
UpdatePeriodChange	The update period of a periodically updated source has changed.

Table 5.1: Classes of sources events

SourceDataUpdate events are accumulated in a repository, storing the changed source and a timestamp corresponding to the instant of the change. A certain time period is chosen, according to the characteristics of the application and the source. Each time this period passes, a quality model for the source is calculated, from the repository data. The calculation of this model for freshness is described in Chapter 4, Section 3.1.1, Model 3, and for accuracy, in Chapter 4, Section 3.1.2.

When an *UpdateFreqChange* event is received by the QMS, it calculates a new model for the corresponding source, with the new λ. This calculation is described in Chapter 4, Section 3.1.1, Model 2.

When an *UpdatePeriodChange* event is received by the QMS, it calculates a new model for the corresponding source, with the new period. This calculation is described in Chapter 4, Section 3.1.1, Model 1.

In all cases, once a new model is calculated, it is stated as the current model for the source (and the corresponding quality factor) and added to the history of the source models, as follows:

- For source *S1*, quality factor *qf1*, and distribution = {<qv_1, $prob_1$>, …, <qv_n, $prob_n$>}, we create a source behavior model:
  - <*qf1*, *S1*, *distribution*>, where *distribution* = {<qv_1, $prob_1$>, …, <qv_n, $prob_n$>}
- We add the model, with name *model1*, such that:
  $((\rho_V (S1))$ (SourceQualityBehaviorModel)) ("model1") = <*qf1*, *S1*, *distribution*>
- The following indicators are calculated:
  - *minimum*, *maximum*, *expectation* and *mode* are calculated from <qv_1, $prob_1$>, …, <qv_n, $prob_n$>
  - *timestamp* = current date and time
- We modify the models history of the source *S1* and quality factor *qf1*, adding the new distribution:
  - *hist1* = ObtainHistory (ρ_V (*S1*) (SourceQualityModelsHistory), *qf1*, *S1*)
  - A new field *indicators* is added to the *distributions set* of *hist1*, *indicators* = < *minimum*, *maximum*, *expectation*, *mode*, *timestamp* >
  - We substitute the current models history of *S1* and *qf1*, by *hist1*.

4.2 Events that are internal to the QMS

At the QMS we generate events that notify a change on a source quality model, a change on the transformation graph or a change on the user quality requirements. These events originate a series of verifications and eventually the generation of other new events. The succession of events may culminate with the notification of the dissatisfaction of user quality requirements, which means that we detected a **relevant** change.

We group these events in three levels, according to their meaning and to how they inter-relate. Level-1 events cause the generation of level-2 events and level-2 events cause the generation of level-3 events. Level-1 events are fired when a change that may affect the DIS has occurred, level-2 events are fired in the cases where DIS quality or DIS required quality has changed, and level-3 ones are fired in the cases where quality requirements are not being satisfied.

In Figure 5.4 we show how each level of events (denoted as L1, L2 and L3) corresponds to a group of changes as previously classified. As can be seen, in the case of source quality, we pay attention to changes that have affected the source quality model, filtering those which have not changed the model.

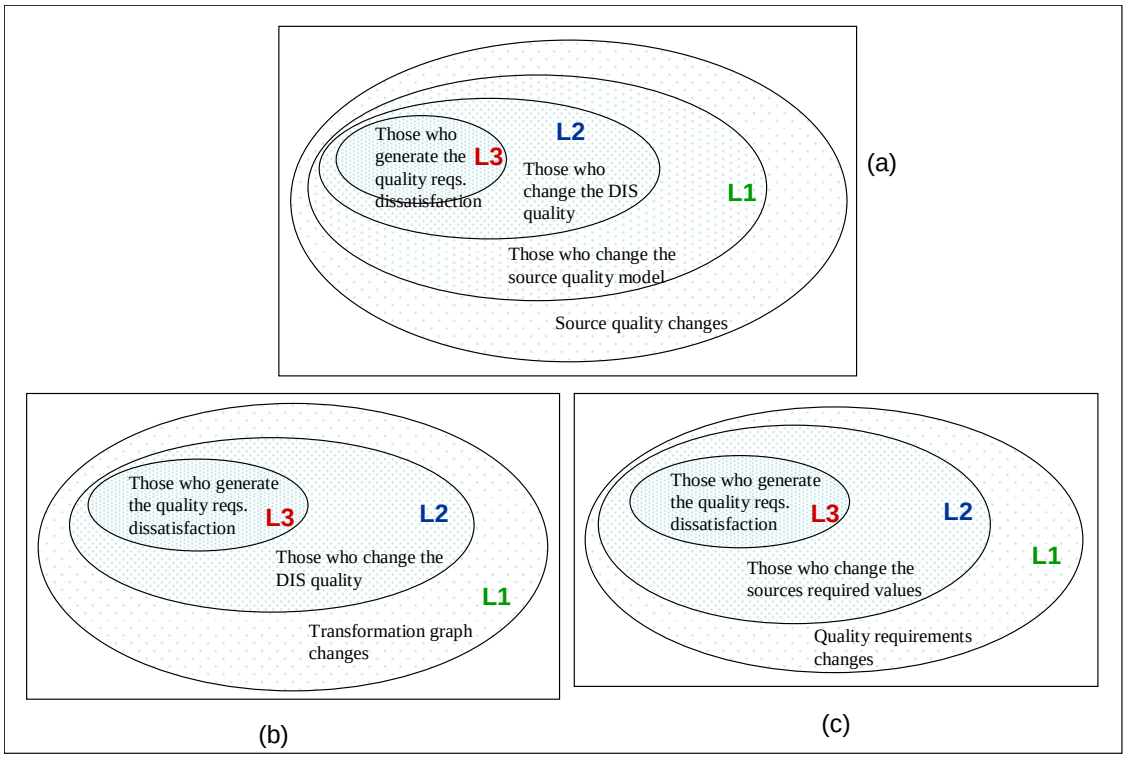


Figure 5.4: Event levels in different types of changes. (a) Source changes classification, (b) Transformation graph changes classification, (c) User quality requirements changes classification.

We define seven classes of events. The events of these classes give alerts notifying the following situations:

- Quality model of a source has changed.
  One of the indicators of the quality model of a source has changed. These indicators are: the maximum, minimum, expectation and mode of the probability distribution of a quality factor in a source.
- The transformation graph of the DIS has changed.
  A change on the structure of the graph or a change on an activity's property (like the cost or the effectiveness) has occurred.
- There was a change on the user quality requirements.
  There was a change on one user quality requirement, an addition or an elimination of a user quality requirement.

- DIS Quality Certainty has changed.
- The degree of satisfaction of quality requirements has changed.
  This means that there was a change that affected the satisfaction of some quality requirements, i.e. there was a change in the relation between a quality value provided by the DIS and the corresponding quality value required by the user.
- There is a required probability that is not being satisfied by the DIS.
  There is at least one quality requirement of type "probability", whose probability value is greater than the DIS Quality Certainty.
- There is a quality requirement that is not being satisfied by the DIS.
  There is at least one quality requirement of type different from "probability", associated to a data target, such that the quality value provided by the DIS in that data target does not satisfy it.

The events classes are presented in Table 5.2, with the level to which they correspond.

Event Level	**Event Name**	**Event Description**
Level 1	QModelChange	The probability distribution of a quality factor has changed in a source.
	TGraphChange	There was a change in the transformation graph of the DIS.
	QReqChange	There was a change in the quality requirements.
Level 2	CertaintyChange	DIS Quality Certainty has changed.
	QSatisfactionChange	The degree of satisfaction of quality requirements has changed.
Level 3	ProbabilityDissatisfaction	Required probability for required quality value is not satisfied by the DIS.
	QValuesDissatisfaction	Required quality value is not satisfied by the DIS.

Table 5.2: Classes of QMS events

With the only objective of showing graphically how the different events generate the creation of others, we take advantage of the idea of collaboration diagram from UML. Figure 5.5 presents a collaboration diagram of the events classes that shows how the events interact. The label "Create(att-vals)" means that the event invokes the creation of the other one (according to the arrow sense) passing the values of the attributes to it.

Each event class has a set of attributes whose values are necessary at the moment of events management, for example, the source that was affected in a QModelChange event. With illustration purposes we show the detailed specification of two of the event classes in Table 5.3, while the complete specification can be read in Appendix III.

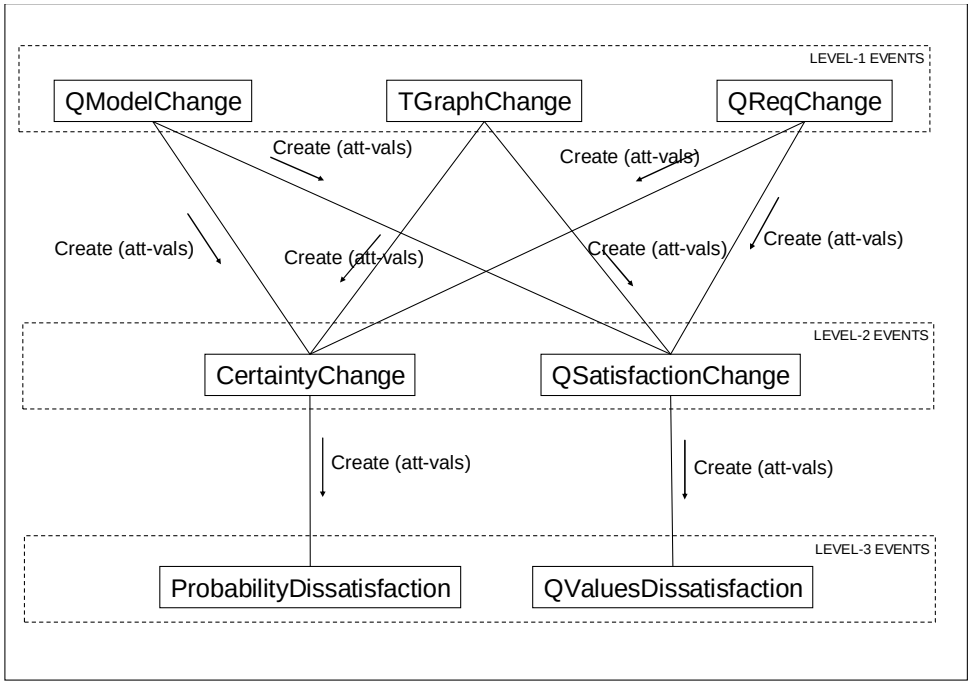


Figure 5.5: Interaction of events

ClassName	Attribute	Attribute Description
QModelChange	QGraph	Quality Graph where the change occurred.
	QFactor	Quality factor corresponding to the changed model.
	SourceName	Name of the source whose quality model has changed.
	ChangedIndicators	Set of model indicators (min, expectation, etc.) that changed.
	Timestamp	Date-time of the change.
CertaintyChange	QGraph	Quality Graph where the change occurred.
	QFactor	Quality factor affected by the change.
	OriginalChange	Change that generated the level-1 event.
	OriginalSource	Source where the change occurred
	GraphChange	Kind of change that occurred on the transformation graph.
	ChangedActivity	Graph activity where a change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the original change.

Table 5.3: Specification of some events

The idea is that events from a level pass information to events from the following level, such that at the end of the chain, if a level-3 event is created, it is able to provide information about the occurred change.

As can be seen in Table 5.3, the attributes of event *CertaintyChange* cover the different possibilities of changes that may occurred when level-1 event was generated. Therefore, some of these attributes will have the NULL value. If the original change was a source quality model change, attributes GraphChange, ActivityChange and ChangedReq are NULL, if it was a transformation graph change, OriginalSource and ChangedReq are NULL, and if it was a user quality requirement change OriginalSource, GraphChange and ActivityChange are NULL.

4.3 Generation and capture of QMS events

Events that are internal to the QMS are generated by different mechanisms.

Each generated event is immediately captured and treated by the *Change Detection Rules*, which are presented in the following section.

Level-1 events are related to changes on objects (models, graph, requirements), and they are generated by the mechanisms that manage them. *QModelChange* events are generated by triggers of the repository that stores the sources quality models. *TGraphChange* and *QReqChange* events are generated by triggers of the database that stores the Quality Graph. In the following we describe the generation of these events:

***QModelChange* events**

Let ind_1 be a 5-uple *indicators* / $ind_1 = < minimum_1, maximum_1, expectation_1, mode_1, timestamp_1 >$ of a *source quality models history*, where $timestamp_1$ is the most recent timestamp of the *distributions* set.

A *QModelChange* event is generated when:

- A new 5-uple *indicators*, $ind_2 = < minimum_2, maximum_2, expectation_2, mode_2, timestamp_2 >$ is added to the *distributions set*, AND
- $minimum_2 \neq minimum_1$ OR $maximum_2 \neq maximum_1$ OR $expectation_2 \neq expectation_1$ OR $mode_2 \neq mode_1$

***TGraphChange* events**

Given a quality graph *G*, an event of this class is generated when a change occurs over one of the following elements:

- the set of activity nodes of *G*
- the set of edges of *G*
- a property of type “feature” associated to an activity node
- a property of type “feature” associated to an edge

Note: An event of this class is generated for each quality factor managed in the system.

***QReqChange* events**

Given a quality graph *G* and its set of target nodes *V*, let *R* be the set of all user quality requirements, which are associated to different target nodes of *V*.

A *QReqChange* event is generated when one of the following occurs:

- in a requirement $r \in R$, $r = <name, qfactor, type, value, prob>$, *value* changes or *prob* changes
- a new requirement is added to R
- a requirement $r \in R$ is eliminated

Level-2 and level-3 events are generated by the *Change Detection Rules*. Level-3 events may also be generated by an execution of the *complete DIS quality verification*, presented in Section 2.

5. Identification of Relevant Changes

The management of the previously presented events allows the Change Detection module of the QMS to identify the relevant changes and to notify them to Quality Repair module, so that it takes the corresponding actions.

We process the events through a set of ECA rules, called *Change Detection Rules*, which are executed when there is a change, they discard the changes that do not affect negatively the DIS, and they notify the relevant changes. These rules process events that are generated at the QMS. They execute different verifications, according to the type of requirements that are affected by the change. Generalizing, some rules check if the occurred change affects existing quality requirements, while other ones check if certain set of quality requirements are being satisfied.

5.1 Change Detection Rules

We use Event Condition Action rules [Elmasri+00] as a tool for specification of events management.

The rules for level-1 events may be triggered if there is a change on a source quality model, on the transformation graph or on the user quality requirements. These rules, considering the occurred change and the user quality requirements that are affected by the change, make other rules be executed through the creation of level-2 events. The rules for level-2 events determine if there are user quality requirements that are not satisfied by the DIS and in that case generate the corresponding level-3 events.

We define a set of rules for each class of event. The following is a general description of the rules.

- Rules for *QModelChange* events.
  We define a rule for each possible type of user quality requirement. In the cases of types different from "probability", each rule verifies if the source model change has modified an indicator that is involved by a requirement of the considered type. If this condition is verified, it creates an event of class *QSatisfactionChange*, passing to it all the requirements of the considered type that involves the changed source. In the case of "probability" type, the rule creates an event of class *CertaintyChange* passing to it all the existing requirements of "probability" type.
- Rules for *TGraphChange* events.
  We define a rule for each possible type of user quality requirement. Each rule verifies if it exists a requirement of the considered type, and in this case it creates an event of class *QSatisfactionChange* or *CertaintyChange*, according to the requirement type, passing to it, all the existing requirements of the considered type.
- Rules for *QReqChange* events.
  We define a rule for each possible type of user quality requirement. Each rule verifies if the changed-requirement's type is the considered in the rule, and in this case it creates an event of class *QSatisfactionChange* or *CertaintyChange*, according to the requirement type, passing to it, all the existing requirements of the considered type.
- Rule for *QSatisfactionChange* events.
  It verifies if the accepted configurations that correspond to the received (through the event) set

of user quality requirements are verified by the sources. If this condition is not verified it creates an event of class *QValuesDissatisfaction*.

- Rule for *CertaintyChange* events.
  It verifies if the DIS Quality Certainty is greater or equal to all the probabilities of the received (through the event) user quality requirements. If this condition is not verified it creates an event of class *ProbabilityDissatisfaction*.

In the following we give a high-level specification of the rules in order to state what they are intended to do.

Change Detection Rules

Rules for *QModelChange* events:

EVENT: *QModelChange*
CONDITION: ∃ r, quality requirement that involves the changed source AND
r is of type "minimum" AND
source minimum has changed
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "minimum" that involves the changed source.

EVENT: *QModelChange*
CONDITION: ∃ r, quality requirement that involves the changed source AND
r is of type "maximum" AND
source maximum has changed
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "maximum" that involves the changed source.

EVENT: *QModelChange*
CONDITION: ∃ r, quality requirement that involves the changed source AND
r is of type "average" AND
source expectation has changed
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "average" that involves the changed source.

EVENT: *QModelChange*
CONDITION: ∃ r, quality requirement that involves the changed source AND
r is of type "frequency" AND
source mode has changed
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "frequency" that involves the changed source.

EVENT: *QModelChange*
CONDITION: ∃ r, quality requirement of type "probability"
ACTION: Create event *CertaintyChange*, with set of requirements of type "probability".

Rules for *TGraphChange* events:

EVENT: *TGraphChange*
CONDITION: ∃ r, quality requirement of type "minimum"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "minimum".

EVENT: *TGraphChange*
CONDITION: ∃ r, quality requirement of type "maximum"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "maximum".

EVENT: *TGraphChange*
CONDITION: ∃ r, quality requirement of type "average"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "average".

EVENT: *TGraphChange*
CONDITION: ∃ r, quality requirement of type "frequency"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "frequency".

EVENT: *TGraphChange*
CONDITION: ∃ r, quality requirement of type "probability"
ACTION: Create event *CertaintyChange*, with set of requirements of type "probability".

Rules for *QReqChange* events:

EVENT: *QReqChange*
CONDITION: Affected requirement is of type "minimum"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "minimum"

EVENT: *QReqChange*
CONDITION: Affected requirement is of type "maximum"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "maximum"

EVENT: *QReqChange*
CONDITION: Affected requirement is of type "average"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "average".

EVENT: *QReqChange*
CONDITION: Affected requirement is of type "frequency"
ACTION: Create event *QSatisfactionChange*, with set of requirements of type "frequency".

EVENT: *QReqChange*
CONDITION: Affected requirement is of type "probability"
ACTION: Create event *CertaintyChange*, with set of requirements of type "probability".

Rule for *QSatisfactionChange* events:

EVENT: *QSatisfactionChange*
CONDITION: Accepted Configurations for the received quality requirements are not satisfied by the sources
ACTION: Create event *QvaluesDissatisfaction.*

Rule for *CertaintyChange* events:

EVENT: *CertaintyChange*
CONDITION: The probability of one of the received quality requirements is greater than the DIS Quality Certainty
ACTION: Create event *ProbabilityDissatisfaction.*

Through the application of these rules we filter changes that are not relevant to the system. For instance, suppose that the quality requirements are only on maximum values, and the source quality model changes, but the maximum value of the source continues being the same. In this case none of the rules are applied and therefore the change is ignored.

There are two filters, which are applied between the events' levels. The first one is between level-1 and level-2 events and filters the changes that do not affect the aspects of the quality considered in the requirements. The second one is between level-2 and level-3 events and filters the changes that do not generate the dissatisfaction of the quality requirements. (See Figure 5.6)

Only if the change leads to the dissatisfaction of a quality requirement, a non-satisfaction event will finally be generated and passed as a notification to another layer of change management. Besides, this notification event will provide a considerable amount of information about the change that is useful for taking the appropriate decisions and actions.

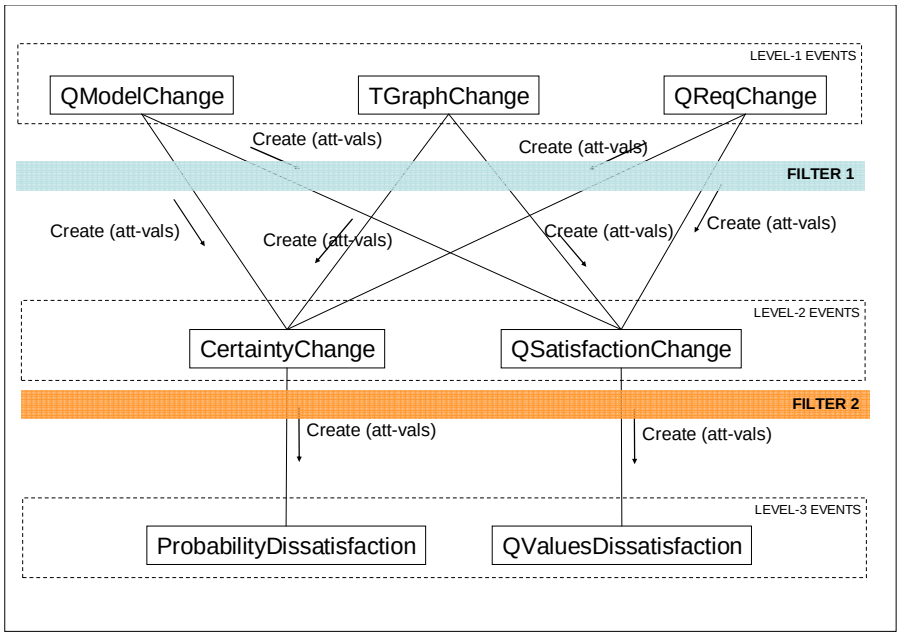


Figure 5.6: Filtering of changes

5.2 Specification of Change Detection Rules

In order to show how the rules are specified, we present, as an example, some of them. The complete specification can be read in Appendix III.

Change Detection Rules

EVENT: e1: *QModelChange*
CONDITION: SelectReqs ('average',
GetCorrespondingRequirements(e1.QGraph, e1.QFactor, e1.SourceName)) ≠ ∅
AND 'expectation' ∈ e1.ChangedIndicators
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = SelectReqs ('average',
GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName))
- OriginalChange = 'QModelChange'
- OriginalSource = e1.SourceName
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, 'probability') ≠ ∅
ACTION: Create event e2: *CertaintyChange*, attribute values:
- QGraph = e1.QGraph
- QFactor = e1.QFactor
- OriginalChange = 'TGraphChange'
- OriginalSource = NULL
- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *QReqChange*
CONDITION: GetType(e1.ChangedReq) = 'maximum'
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph,
GetQFactor(e1.ChangedReq), 'maximum')
- OriginalChange = 'QReqChange'
- OriginalSource = NULL
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

EVENT: e1: *CertaintyChange*
CONDITION: NOT Quality_Certainty_Verification (e1.QGraph, e1.QFactor)
ACTION: Create event e2: *ProbabilityDissatisfaction*, attribute values:
- QGraph = e1.QGraph
- QFactor = e1.QFactor

- OriginalChange = e1. OriginalChange
- OriginalSource = e1. OriginalSource
- GraphChange = e1. GraphChange
- ChangedActivity = e1. ActivityChange
- ChangedReq = e1. ChangedReq
- Timestamp = e1.Timestamp

Description of used auxiliary functions:

Function *SelectReqs* selects from a set of user quality requirements, those whose type is the given one.

Function *GetCorrespondingRequirements* obtains for a source, the quality requirements of certain quality factor, that involve it.

Function *Get_Requirements* obtains, given a quality graph and a quality factor, all the requirements of certain type (which are associated to data targets of the given graph).

Function *GetType* returns the type of a user quality requirement.

Function *GetQFactor* returns the quality factor of a user quality requirement.

Function *Quality_Certainty_Verification* receives a quality graph and a quality factor, and verifies if the DIS Quality Certainty satisfies the user quality requirements (specified in Section 2).

6. Example

In this section we present an example that shows a possible scenario in a DIS, and how two different situations of quality change are detected.

This example not only covers problems of this chapter but also of Chapter 4, since it is intended to show the complete situation and resolution. This example scenario is based on one presented in [Peralta-06] and parts of it have already been presented in some examples of Chapter 4.

6.1 The DIS

Consider a DIS built for retrieving meteorological information, whose quality graph is illustrated in Figure 5.7. Quality factor freshness is evaluated. There are three source relations: S_1 with real time satellite meteorological predictions, S_2 which is a dissemination database updated once a day and S_3 with information about climatic sensors which is published once an hour. S_1 is updated irregularly, S_2 is updated periodically, each 24 hours, and S_3 is updated periodically with period 1 hour.

The goal of the system is to provide fresh meteorological information to solve three types of queries: T_1 (historical information about climate alerts), T_2 (aggregated data about predictions) and T_3 (detailed data about climate measurements). Users require that freshness of retrieved data does not exceed 72, 48 and 2 hours respectively. Additionally, they require that the average values obtained in each query are: 60, 42 and 2 hours respectively.

The DIS is composed of nine activities that process the information performing the extraction, filtering, integration and aggregation of data. Figure 5.7 shows the quality graph; processing costs and synchronization delays are expressed in hours. Activity A5 executes every 12 hours, materializing the produced data; for that reason, synchronization delay with activity A6 is 12 hours. Analogously, activity A6 materializes data every 7 hours. The other activities execute coordinated,

one immediately after its predecessor (delay 0). Activities A7, A8 and A9 execute for each user query and activities A1, A2, A3 (extraction activities) and A4, execute when successor activities ask them for data.

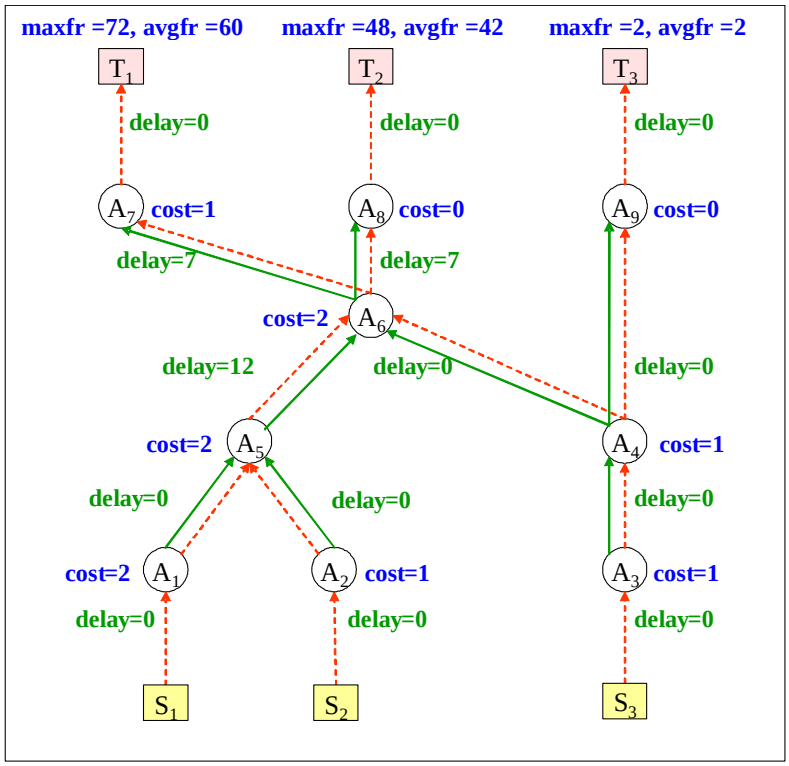


Figure 5.7: DIS Quality Graph

6.2 Sources' Models

We build the quality behavior models of the sources, using as discretization unit: 1 hour.

Source S1:

Since S1 is updated irregularly, we must estimate the probability distribution from the *SourceDataUpdate* events repository (as it was done in Chapter 4, Example 4.2). Table 5.4 is a portion of the events repository, where Date-time is the date and time in the format "dd/mm-hh", where time is represented in hours.

Source	Date-time
S1	2/11-2
S1	2/11-3
S1	2/11-5
S1	2/11-8
S1	2/11-9
S1	2/11-10
S1	2/11-12
S1	2/11-14

S1	2/11-18
S1	2/11-21
S1	2/11-22

Table 5.4: SourceDataUpdate events repository

From the events repository we calculate the freshness that held at each time point (see Table 5.5), considering the first time point as 2/11-2, the first timestamp of the events repository, and a time step of 1 hour.

Source	time point	Freshness
S1	2/11-2	0
S1	2/11-3	0
S1	2/11-4	1
S1	2/11-5	0
S1	2/11-6	1
S1	2/11-7	2
S1	2/11-8	0
S1	2/11-9	0
S1	2/11-10	0
S1	2/11-11	1
S1	2/11-12	0
S1	2/11-13	1
S1	2/11-14	0
S1	2/11-15	1
S1	2/11-16	2
S1	2/11-17	3
S1	2/11-18	0
S1	2/11-19	1
S1	2/11-20	2
S1	2/11-21	0
S1	2/11-22	0

Table 5.5: Calculated freshness

Then we calculate the relative frequencies of the freshness values (see Table 5.6). For each freshness value, the relative frequency is the number of occurrences of the value divided by 21, which is the quantity of hours chosen as our sample.

Source	Freshness value	Relative frequency
S1	0	11/21
S1	1	6/21
S1	2	3/21
S1	3	1/21

Table 5.6: Relative frequencies of freshness values

Let X be the RV that represents the freshness of source S1. The following is the probability distribution of X:

P(X=0) = 0.52
P(X=1) = 0.28
P(X=2) = 0.14
P(X=3) = 0.04

In order to add the model to the source information of the QMS, we follow these steps:

- We create the source behavior model:

  <freshness, S1, *distribution*>, where

  distribution = {<0, 0.52>, <1, 0.28>, <2, 0.14>, <3, 0.04>}

- We add this model, with name "fresh-meteo", such that:

  ρ_V (S1) (SourceQualityBehaviorModel) ("fresh-meteo") = <freshness, S1, *distribution*>

- We calculate indicators:

  minimum = 0
  maximum = 3
  expectation = Σ_x xp(x) = 0*0.52 + 1*0.28 + 2*0.14 + 3*0.04 = 0.68
  mode = 0

- We modify the models history of the source S1 and quality factor freshness, adding the new distribution
  <0, 3, 0.68, 0, 2/11-22>, where 2/11-22 is the current date-time.

Source S2:

This source is updated each 24 hours.

- We create the source behavior model:

  <freshness, S2, *distribution*>, where

  distribution = {<0, 0.04>, <1, 0.04>, <2, 0.04>, <3, 0.04>, …, <23, 0.04>}

- We add this model, with name "fresh-meteo", such that:

  ρ_V (S2) (SourceQualityBehaviorModel) ("fresh-meteo") = <freshness, S2, *distribution*>

- We calculate indicators:

  minimum = 0
  maximum = 23
  expectation = Σ_x xp(x) = 11

- We modify the models history of the source S2 and quality factor freshness, adding the new distribution
  <0, 23, 11, NULL, current-date-time>

Source S3:

This source is updated each hour. As our precision is 1 hour, we consider this source as always fresh.

- We create the source behavior model:

<freshness, S3, *distribution*>, where

distribution = {<0, 1>}

- We add this model, with name “fresh-meteo”, such that:

ρ_V (S3) (SourceQualityBehaviorModel) (“fresh-meteo”) = <freshness, S3, *distribution*>

- We calculate indicators:

minimum = 0
maximum = 0
expectation = Σ_x xp(x) = 0
mode = 0

- We modify the models history of the source S2 and quality factor freshness, adding the new distribution
<0, 0, 0, 0, current-date-time>

6.3 User quality requirements

We assign names to the requirements: Rmax1, Ravg1, Rmax2, Ravg2, Rmax3, Ravg3, such that:

ρ_V (T1) (UserQualityRequirement) (Rmax1) = < freshness, maximum, 72, NULL >

ρ_V (T2) (UserQualityRequirement) (Rmax2) = < freshness, maximum, 48, NULL >

ρ_V (T3) (UserQualityRequirement) (Rmax3) = < freshness, maximum, 2, NULL >

ρ_V (T1) (UserQualityRequirement) (Ravg1) = < freshness, average, 60, NULL >

ρ_V (T2) (UserQualityRequirement) (Ravg2) = < freshness, average, 42, NULL >

ρ_V (T3) (UserQualityRequirement) (Ravg3) = < freshness, average, 2, NULL >

6.4 Initial DIS Quality Verification

We apply the algorithm specified in Section 2.

1) We verify the quality satisfaction for “maximum” user requirements:

- Requirements = { <T1, Rmax1>,
<T2, Rmax2>,
<T3, Rmax3> }

- Accepted Configurations, named: “ac_maximum”:

gp(AcceptedConfigurations) (ac_maximum) = < {<T1, Rmax1>, <T2, Rmax2>, <T3, Rmax3>},
{<restriction1, restriction2, restriction3>} > , where:

restriction1 = <freshness, S1, “≤”, 23>
restriction2 = <freshness, S2, “≤”, 24>
restriction3 = <freshness, S3, “≤”, 0>

The calculation of accepted configurations is done through the algorithm of Chapter 4, Section 4.1. We briefly show how it is done.

For the propagation we use the following formula, for an activity A:

Freshness(A) = Max (Freshness(A_1) + Delay(A_1,A), …, Freshness(A_n) + Delay(A_n,A)) + Cost(A), where A_1, …, A_n are A predecessors

For example, for T1 and requirement of maximum, the propagation is done as follows:

fr(T1) ≤ 72
fr(T1) = fr(A7) = fr(A6) + delay(A6, A7) + cost(A7) = fr(A6) + 7 + 1 = fr(A6) + 8
fr(A6) = Max (fr(A5) + delay(A5, A6), fr(A4) + delay(A4, A6)) + cost(A6) =
Max (fr(A5) + 12, fr(A4)) + 2
fr(A5) = Max (fr(A1) + delay(A1, A5), fr(A2) + delay(A2, A5)) + cost(A5) =
Max (fr(A1), fr(A2)) + 2
fr(A4) = fr(A3) + delay(A3, A4) + cost(A4) = fr(A3) +1
fr(A1) = fr(S1) + delay(S1, A1) + cost(A1) = fr(S1) + 2
fr(A2) = fr(S2) + delay(S2, A2) + cost(A2) = fr(S2) + 1
fr(A3) = fr(S3) + delay(S3, A3) + cost(A3) = fr(S3) + 1

fr(T1) = Max [Max (fr(S1) + 2, fr(S2) + 1) + 2 + 12, fr(S3) + 1 +1] + 2 + 8
Max [Max (fr(S1) + 2, fr(S2) + 1) + 14, fr(S3) + 2] + 10 ≤ 72
Max (fr(S1) + 2, fr(S2) + 1) + 14 + 10 ≤ 72
Max (fr(S1) + 2, fr(S2) + 1) ≤ 48
fr(S1) ≤ 46
fr(S2) ≤ 47
fr(S3) + 2 + 10 ≤ 72
fr(S3) ≤ 60

Table 5.7 shows the propagated values form the requirements to the sources. For each source and each data target the table presents the propagated value for the requirement of maximum.

source	requirement	maximum
S1	T1	46
S1	T2	23
S2	T1	47
S2	T2	24
S3	T1	60
S3	T2	37
S3	T3	0

Table 5.7: Propagated values for each user quality requirement

Finally, the smallest value is selected for the restriction of each source.

- We verify the quality satisfaction of the requirements of maximum (algorithm specified in Chapter 4, Section 4.3):

For each source restriction we verify if it is satisfied. Maxima of the sources models were obtained in Section 6.2.

restriction1 = <freshness, S1, "≤", 23>

maximum of S1 = 3

3 ≤ 23, verifies

Analogously with restriction2 and restriction3:

23 ≤ 24, verifies
0 ≤ 0, verifies

Maximum requirements satisfaction = TRUE

2) We verify the quality satisfaction for "average" user requirements:

- Requirements = { <T1, Ravg1>,
<T2, Ravg2>,
<T3, Ravg3> }

- Accepted Configurations, named: "ac_average", are calculated analogously to the ones for requirements of maximum.

gp(AcceptedConfigurations) (ac_average) = < {<T1, Ravg1>, <T2, Ravg2>, <T3, Ravg3>},
{<restriction1, restriction2, restriction3>} > , where:
restriction1 = <freshness, S1, "≤", 17>
restriction2 = <freshness, S2, "≤", 18>
restriction3 = <freshness, S3, "≤", 0>

- We verify the quality satisfaction of the requirements of average:

For each source restriction we verify if it is satisfied. Expectations of the sources models were obtained in Section 6.2.

restriction1 = <freshness, S1, "≤", 17>
expectation of S1 = 0.68

0.68 ≤ 17, verifies

Analogously with restriction2 and restriction3:

11 ≤ 18, verifies
0 ≤ 0, verifies

Average requirements satisfaction = TRUE

We conclude that DIS quality is being verified.

6.5 Detection of First Change

Occurred change:

The property *cost* of the activity node A2 has changed (see Figure 5.8).

Before: ρ_V(A2) (Cost) (“cost”) = 1
Now: ρ_V(A2) (Cost) (“cost”) = 4

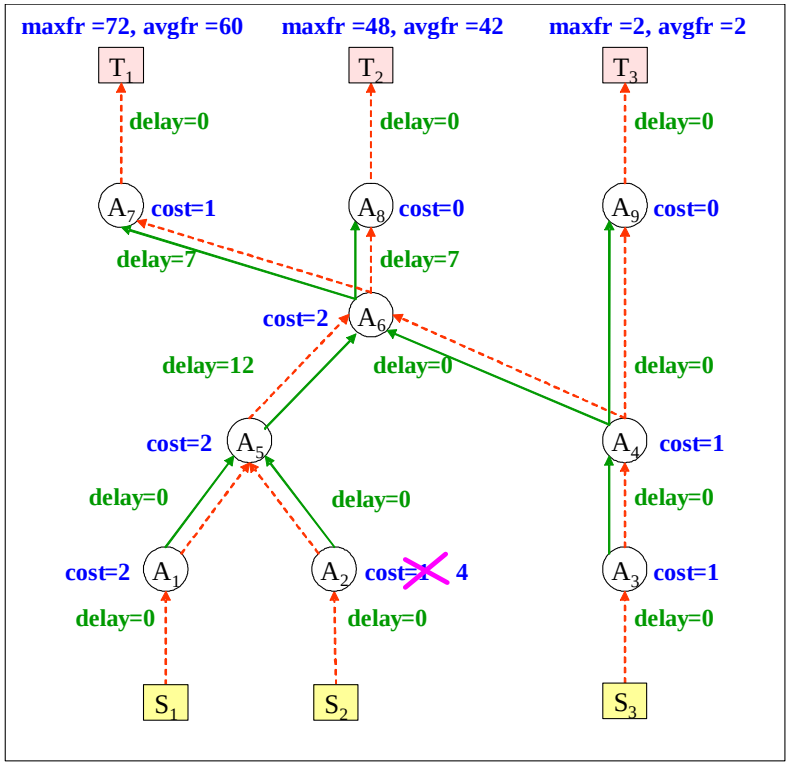


Figure 5.8: Transformation graph change

Event **TGraphChange** is created, with attributes:

QGraph = “MeteoGraph”
QFactor = freshness
Type = activity-cost
ActivityName = A2
Timestamp = 5/11-3

Detection Rules application:

The following rule is applied:

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, ‘maximum’) ≠ ∅
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements =Get_Requirements (e1.QGraph, e1.QFactor, ‘maximum’)
- OriginalChange = “TGraphChange”
- OriginalSource = NULL

- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

Event **QSatisfactionChange** is created, with attributes:

QGraph = "MeteoGraph"
Requirements = { <T1, Rmax1>, <T2, Rmax2>, <T3, Rmax3> }
OriginalChange = "TGraphChange"
OriginalSource = NULL
GraphChange = activity-cost
ChangedActivity = A2
ChangedReq = NULL
Timestamp = 5/11-3

Is the following rule applied ?

EVENT: e1: *QSatisfactionChange*
CONDITION: NOT QualitySatisfaction (e1.QGraph,
Accepted_Conf_Calculation (e1.QGraph,
GetQFactor(e1.Requirements), e1.Requirements),
GetQFactor(e1.Requirements),
GetType(e1.Requirements))
ACTION: Create event e2: *QValuesDissatisfaction*, attribute values:
- QGraph = e1.QGraph
- Requirements = e1.Requirements
- OriginalChange = e1. OriginalChange
- OriginalSource = e1. OriginalSource
- GraphChange = e1. GraphChange
- ChangedActivity = e1. ChangedActivity
- ChangedReq = e1. ChangedReq
- Timestamp = e1.Timestamp

The condition is verified:

- Requirements = { <T1, Rmax1>, <T2, Rmax2>, <T3, Rmax3> }
- Accepted Configurations, named: "ac_maximum":

gp(AcceptedConfigurations) (ac_maximum) = < {<T1, Rmax1>, <T2, Rmax2>, <T3, Rmax3>},
{<restriction1, restriction2, restriction3>} > , where:
restriction1 = <freshness, S1, "≤", 23>
restriction2 = <freshness, S2, "≤", **21**>
restriction3 = <freshness, S3, "≤", 0>

The restriction over S2 (restriction2) has changed.

- QualitySatisfaction:

The sources models have not changed, S2 maximum = 23, we verify S2 satisfaction:

23 ≤ 21, does not verify

QualitySatisfaction = FALSE

The condition of the rule is satisfied.

Event **QValuesDissatisfaction** is created, with attributes:

QGraph = “MeteoGraph”
Requirements = { <T1, Rmax1>, <T2, Rmax2>, <T3, Rmax3> }
OriginalChange = “TGraphChange”
OriginalSource = NULL
GraphChange = activity-cost
ChangedActivity = A2
ChangedReq = NULL
Timestamp = 5/11-3

The following rule is also applied:

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, ‘average’) ≠ ∅
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, e1.QFactor, ‘average’)
- OriginalChange = ‘TGraphChange’
- OriginalSource = NULL
- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

Event **QSatisfactionChange** is created, with attributes:

QGraph = “MeteoGraph”
Requirements = { <T1, Ravg1>, <T2, Ravg2>, <T3, Ravg3> }
OriginalChange = “TGraphChange”
OriginalSource = NULL
GraphChange = activity-cost
ChangedActivity = A2
ChangedReq = NULL
Timestamp = 5/11-3

The condition of the rule that corresponds to this event is verified:

- Analogously to the case for “maximum” requirements, accepted configurations have changed.

gp(AcceptedConfigurations) (ac_average) = < {<T1, Ravg1>, <T2, Ravg2>, <T3, Ravg3>}, {<restriction1, restriction2, restriction3>} > , where:
restriction1 = <freshness, S1, “≤”, 17>
restriction2 = <freshness, S2, “≤”, **15**>
restriction3 = <freshness, S3, “≤”, 0>

The restriction over S2 (restriction2) has changed.

- QualitySatisfaction:

The sources models have not changed, S2 average = 11, we verify S2 satisfaction:

11 ≤ 15, verifies

QualitySatisfaction = TRUE

No more rules are applied.

In summary, one relevant change was detected, and event **QValuesDissatisfaction** was created with all the relative information.

6.6 Detection of Second Change

Occurred change:

The source quality model of S1 is re-calculated from the repository data (which contains source data updates). Therefore the new model is stored as the current one:

ρ_V (S1) (SourceQualityBehaviorModel) (“fresh-meteo”) = <freshness, S1, *distribution*>, where

distribution = {<0, 0.25>, <1, 0.12>, <2, 0.07>, <3, 0.02>, …, <24, 0.02>}

The indicators are calculated:

minimum = 0
maximum = 24
expectation = Σ_x xp(x) = 6
mode = 0

The models history of the source S1 and freshness is modified, the new distribution, <0, 24, 6, 0, 5/11-3> is added, where 5/11-3 is the current date-time.

Now:

ρ_V (S1) (SourceQualityModelHistory) (“fresh-meteo”) = <freshness, S1, *distributions*>, where

distributions = { **<0, 24, 6, 0, 5/11-3>**, <0, 3, 0.68, 0, 2/11-22> }

As a consequence of the fact that there are two indicators (maximum and expectation) that have changed their values, event **QModelChange** is created, with attributes:

QGraph = “MeteoGraph”
QFactor = freshness
SourceName = S1
ChangedIndicators = {maximum, expectation}
Timestamp = 5/11-3

Detection Rules application:

The following rule is applied, since the condition is satisfied:

EVENT: e1: *QModelChange*
CONDITION: SelectReqs (‘maximum’,
GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName)) ≠ ∅
AND ‘maximum’ ∈ e1.ChangedIndicators
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = SelectReqs (‘maximum’,
GetCorrespondingRequirements(e1.QGraph, e1.QFactor,
e1.SourceName))
- OriginalChange = ‘QModelChange’
- OriginalSource = e1.SourceName
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

Event **QSatisfactionChange** is created, with attributes:

QGraph = “MeteoGraph”
Requirements = { <T1, Rmax1>, <T2, Rmax2> } // Note that Rmax3 does not involve S1.
OriginalChange = “QModelChange”
OriginalSource = S1
GraphChange = NULL
ChangedActivity = NULL
ChangedReq = NULL
Timestamp = 5/11-3

Is the following rule applied ?

EVENT: e1: *QSatisfactionChange*
CONDITION: NOT QualitySatisfaction (e1.QGraph,
Accepted_Conf_Calculation (e1.QGraph,
GetQFactor(e1.Requirements), e1.Requirements),
GetQFactor(e1.Requirements),
GetType(e1.Requirements))
ACTION: Create event e2: *QValuesDissatisfaction*, attribute values:
- QGraph = e1.QGraph
- Requirements = e1.Requirements
- OriginalChange = e1. OriginalChange
- OriginalSource = e1. OriginalSource
- GraphChange = e1. GraphChange
- ChangedActivity = e1. ChangedActivity
- ChangedReq = e1. ChangedReq
- Timestamp = e1.Timestamp

The condition is verified:

- Requirements = { <T1, Rmax1>, <T2, Rmax2> }
- Accepted Configurations, named: “ac_maximum”:

  gp(AcceptedConfigurations) (ac_maximum) = < {<T1, Rmax1>, <T2, Rmax2>}, {<restriction1, restriction2>} > , where:

  restriction1 = <freshness, S1, “≤”, 23>
  restriction2 = <freshness, S2, “≤”, 24>

  The restrictions over S1 and S2 have not changed.

- QualitySatisfaction:

  S1 quality model has changed, S1 maximum = 24, we verify S1 satisfaction:

  24 ≤ 23, does not verify

  QualitySatisfaction = FALSE

The condition of the rule is satisfied.

Event **QValuesDissatisfaction** is created, with attributes:

QGraph = “MeteoGraph”
Requirements = { <T1, Rmax1>, <T2, Rmax2> }
OriginalChange = “QModelChange”
OriginalSource = S1
GraphChange = NULL
ActivityChange = NULL
ChangedReq = NULL
Timestamp = 5/11-3

The following rule is also applied:

EVENT: e1: *QModelChange*
CONDITION: SelectReqs (‘average’,
GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName)) ≠ ∅
AND ‘expectation’ ∈ e1.ChangedIndicators
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = SelectReqs (‘average’,
  GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName))
- OriginalChange = ‘QModelChange’
- OriginalSource = e1.SourceName
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

Event **QSatisfactionChange** is created, with attributes:

QGraph = “MeteoGraph”
Requirements = { <T1, Ravg1>, <T2, Ravg2> } // Note that Ravg3 does not involve S1.
OriginalChange = “QModelChange”
OriginalSource = S1
GraphChange = NULL
ChangedActivity = NULL
ChangedReq = NULL
Timestamp = 5/11-3

The condition of the rule that corresponds to this event is verified:

- Analogously to the case for “maximum” requirements, the restrictions over S1 and S2 have not changed. Restriction over S1: < freshness, S1, “≤”, 17>

- QualitySatisfaction:

S1 quality model has changed, S1 expectation = 6, we verify S1 satisfaction:

6 ≤ 17, verifies

QualitySatisfaction = TRUE

No more rules are applied.

In summary, one relevant change was detected, and event **QValuesDissatisfaction** was created with all the relative information.

6.7 Example Conclusion

In this example we present a DIS that provides meteorological information, which is constituted by 3 data sources, a transformation graph, and 3 data targets. It has some freshness requirements associated to its data targets; each data target has one “maximum” requirement and one “average” requirement.

We first calculate the sources quality models from the information we have about the sources updates (applying the techniques presented in Chapter 4). Source S1’s model is calculated obtaining the probability distribution through the calculation of relative frequencies. Sources S2 and S3 are updated periodically, so their quality models are calculated from the respective update periods.

Then we verify the quality of the DIS in the initial state (applying mechanism presented in Section 2). The quality is correct, since all the quality requirements are being satisfied.

We then study two different quality changes that may occur, and how the QMS would behave applying the proposed change-detection mechanism, in each alternative case.

The first considered change is a change in the cost of an activity of the transformation graph. This change causes the creation of an event *TGraphChange*. A rule for this event is applied, which considers the “maximum” requirements, and an event *QSatisfactionChange* is created. A rule for this event is applied and an event *QValuesDissatisfaction* is created. This event is the alert that notifies that **a relevant quality change has occurred**. The event contains information about the change, such as the graph activity that has changed and the requirements that are not being satisfied. The same original change also causes the application of another rule that considers the “average” requirements, generating the creation of another *QSatisfactionChange* event. However, this event has no consequences. Figure 5.9 shows the flow of the generated events.

The second considered change is a change in the quality behavior model of source S1. This change causes the creation of an event *QModelChange*. Analogously to the first change, **the change is detected as relevant**, as a consequence of the non satisfaction of "maximum" requirements. Figure 5.9 shows the flow of the generated events.

In summary, in the first change case, a notification of relevant change is generated, informing that there was a change in the cost of the activity A2 of the transformation graph, that the requirements of type "maximum" are not satisfied, and that the change occurred on 5/11, at 3. In the second change case, a notification of relevant change is generated, informing that there was a change in the quality model of source S1, that the "maximum" requirements of target T1 and target T2 are not satisfied, and that the change occurred on 5/11, at 3.

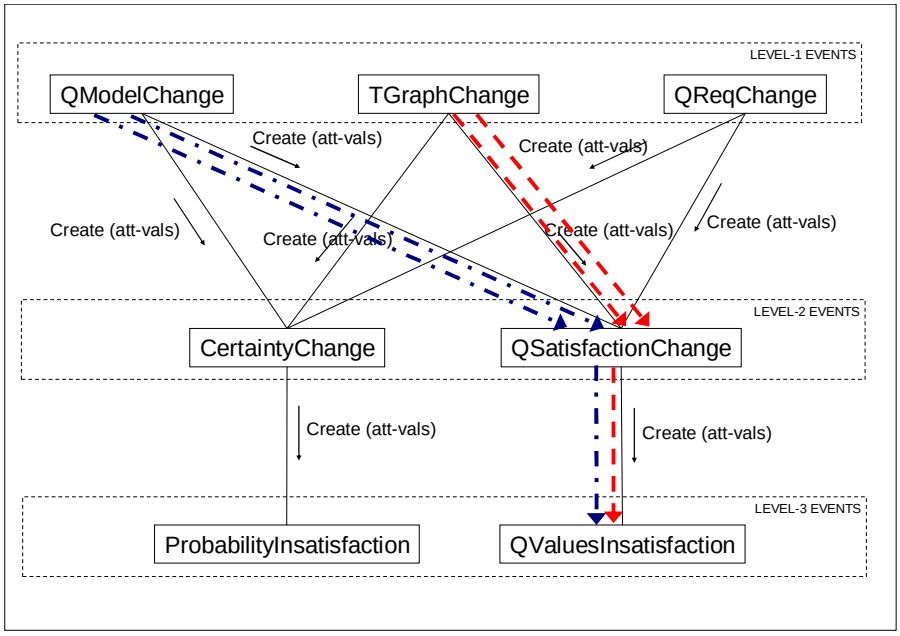


Figure 5.9: Events flow

7. Summary

This chapter focuses on the problem of detecting changes that affect the quality of the DIS.

The considered changes are changes on sources quality, changes on the DIS transformation process and changes on user quality requirements.

We propose a mechanism that allows the QMS to detect that a change has occurred. In the case of source quality changes we assume that certain information about source updates can be obtained, either through an alert provided by the source itself, or analyzing the extracted information. In the case of changes on the transformation or on user quality requirements, we assume that events are automatically triggered from updates on the system metadata.

Once the QMS realizes that a change has occurred, the proposed mechanism deals with the information about the change and about the DIS current state, in order to determine if the change is relevant to the system quality or not. In order to achieve this, a set of events are defined and managed, such that each change passes through two filters, which decide if the change is notified to the quality repair module or it is discarded. The events management is implemented through ECA rules, which we call Change Detection Rules.

Different kinds of events are defined and managed: (i) events that are generated when a change has occurred on a source, (ii) events that are generated when a change has occurred on the quality model of a source, on the DIS transformation process, or on a user quality requirement, (iii) events that are generated when certain conditions related to the occurred change and to the DIS state, are verified. Events from group (i) cause the generation of some of the events from group (ii). Events from (ii) cause the generation of events from group (iii). Events from group (i) are generated by the sources, events from group (ii) are generated by the QMS through metadata triggers, and events from group (iii) are generated by the QMS, specifically by the Change Detection Rules.

During quality changes detection, partial verifications of DIS quality are done, minimizing the workload of the QMS. This chapter also presents a mechanism for doing a complete DIS quality verification, that is, for evaluating if a DIS is satisfying all the existing requirements or not. This is a basic verification, which considers one by one the groups of requirements, grouped by type, and verifies if they are satisfied by the DIS. Variations of this algorithm may be implemented, for example, individualizing each requirement satisfaction, or showing the quality probability distribution of the DIS.

An example is presented in detail, showing the modeling of the sources quality and all the process of relevant change detection. This example enables to see the application of the mechanism to concrete cases helping to better comprehend the proposal.

The proposed mechanism has two main characteristics that give it effectiveness and efficiency. On one hand, it works with a preventive strategy, since it manages source quality model changes instead of changes on punctual values, and it also manages requirements that include probabilities, expected values, and modes (most probable values). On the other hand, it absorbs many changes that do not affect DIS quality, avoiding unnecessary work.

One aspect of the mechanism that could be improved, achieving even more efficiency, is the verification of some of the rules conditions. For example, in the case of transformation-graph change, the verification is done over all the user quality requirements and sources, while it may be done only over the requirements and sources that are involved with the change. For doing this, it is necessary to calculate, given an activity or an edge of the graph, all the requirements and sources that are “connected” to it.

It is important to note that the detection of sources quality changes is done based on past events, since we work with the model of the source quality. This specially happens in the case of models that are built from statistical information. This characteristic has important advantages that have already been commented, but at the same time it has the disadvantage of putting the DIS at the risk of suffering punctual changes that generate the requirements dissatisfaction, which are detected later.

Finally, we want to remark that the solutions proposed in this chapter are totally independent from the quality factor that is considered. The mechanism is applicable to any quality factor, existing the possibility of extending the information passed through the events with information particular to the factor.

CHAPTER 6. DIS QUALITY REPAIR

Quality of our Data Integration System has changed. Now it is not good enough. What should we do?

1. Introduction

We consider that DIS quality is acceptable if it satisfies all user quality requirements, and otherwise it is unacceptable. When the DIS suffers a change that generates the dissatisfaction of user quality requirements, its quality becomes unacceptable. For these cases, we propose to search for actions to repair DIS quality, which means to take DIS quality to an acceptable state.

In this chapter we first present an overview of the possible actions that may be applied to the DIS and sources in order to repair quality. We analyze which actions have sense in the different cases of change. Secondly, we present our proposal for DIS quality repair by the QMS.

In the beginning of our analysis we base ourselves on a problem that seems to be near ours: source schema evolution in information systems with multiple sources. We use as starting point the analysis of how to manage source schema changes in these systems. From that point we find an analogy, when we consider the different changes that affect DIS quality, between the actions we may do for repairing DIS quality and the actions that may be done for propagating source schema evolution. After this initial analysis we arrive to a classification of the repairing actions. We group them according to which part of the DIS they modify and according to the kind of modifications they apply.

Among the possible actions for repairing DIS quality, there are some that modify DIS-elements' properties and others that modify DIS design, i.e. how it combines data and which sources it uses. We intend to propose modifications that affect as less as possible the design of the DIS. We study the different modifications that may be done in order to repair DIS quality in the different contexts and cases of change, for freshness factor and for accuracy factor. In each case we observe how modifications on different DIS elements affect DIS quality. In addition, we analyze how freshness behavior affects accuracy, and vice versa.

There are many different actions that can be applied to improve DIS quality, such as reducing processing costs, restructuring the data transformation graph, adding new tasks, etc. The greatest complexity resides in determining which actions should be applied in order to recover DIS quality from changes that damaged it.

Our goal is to give to the user (DIS designer or administrator) suggestions to repair DIS quality. In our proposal, the QMS is capable of analyzing what has happened taking into account all the available information, and creating a list of possible actions, each of which would take the DIS to an acceptable quality state. This list is a ranking that goes from the most recommended action to the least recommended one.

The QMS uses the following information in order to repair DIS quality: (i) the event received from the Change Detection process, which notifies that a relevant change has occurred and provides information about it, and (ii) the statistical information about DIS properties and quality, which is maintained at the system. The QMS analyses the situation of changes at the DIS taking into account the change that generated the event and the statistics or historical information, arriving to an interpretation. This interpretation is later used to construct the previously mentioned list of repairing actions as well as any warnings that are given when necessary. The repairing actions can be

expressed in terms of the improvement actions, proposed in [Peralta-06], which are basic actions to apply over the quality graph.

The outline of this chapter is as follows. Section 2 presents an analysis and overview of the actions that may be applied to repair DIS quality, Section 3 presents our proposal for quality repair in the QMS, Section 4 presents an example, and Section 5 presents the summary of the chapter.

2. Quality Repairing Actions

There is a wide range of different actions that may be carried out in order to repair DIS quality. In this section we intend to visualize all these possible actions and organize them in a taxonomy.

While analyzing our problem and searching for similar problems that may be already addressed and solved, we find that the problem of source schema evolution in integration systems presents some common aspects with ours. An exhaustive comparison of both problems is presented in Appendix 1 of this document.

2.1 Inspiration in Previously Studied Problem

Taking advantage of the existing similarity between our problem and the problem of source schema evolution in DIS, we use the latter as an inspiration for starting our analysis.

We take the idea of "change propagation", which means calculating how the source change affects the DIS, and includes the possibility of modifying different DIS components in order to avoid the change in the data targets. The similarity consists on the actions that may be carried out in order to manage the source change, minimizing the consequences in the results given to the DIS users.

As we are talking about schema evolution, for simplicity, we suppose a DIS that has a pre-defined integrated schema. Generalizing, we synthesize source schema evolution situations in the following three cases:

(i) Source schema changes are propagated to the integrated schema generating changes on this schema and also changes on the transformation process. Figure 6.1 shows an example of this situation. There are three sources providing data to the relation *Doctors*. In source *S3* the relation *Symptoms* is deleted and this change causes the deletion of the attribute *common_symptoms* from *Doctors*, and its associated transformations.

(ii) Due to a source schema change, the transformation is modified in a way that it absorbs the changes, and the integrated schema is not modified. Figure 6.2 shows an example of this situation. In this case relation *Patients* of source *S2* is changed and must be eliminated from the system. This causes the modification of the transformation and the elimination of source *S2* from the system (including relation *Treatments*). The schema of the relation *Doctors* is not modified.

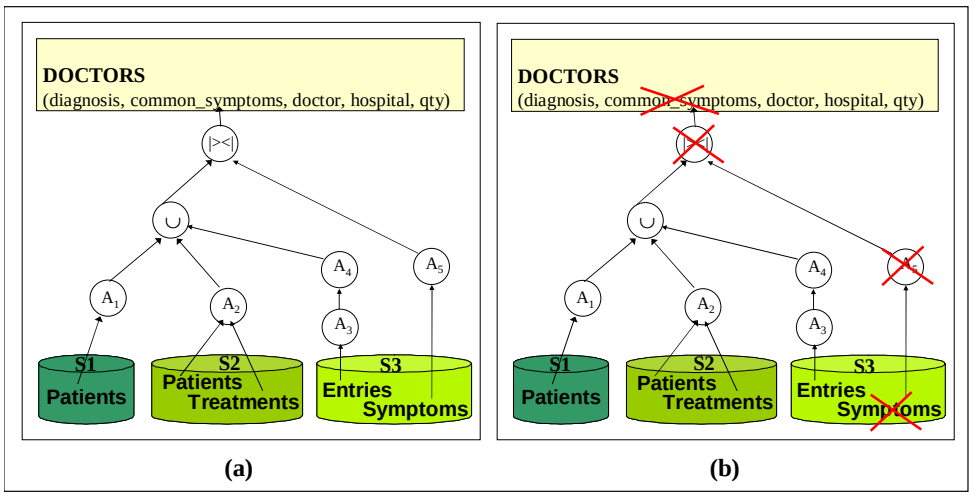


Figure 6.1: Source change propagation to integrated schema. (a) Before change. (b) After change.

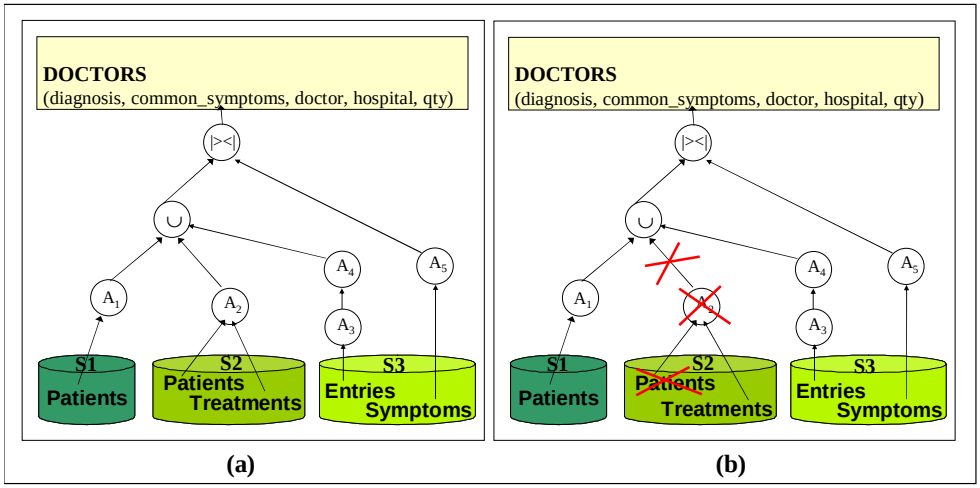


Figure 6.2: Source change propagation, where transformation absorbs the change. (a) Before change. (b) After change.

(iii) A source schema change causes a change in the schema of other source relation. Figure 6.3 shows an example of this situation. In this example the attribute *hospital* of relation *S1.Doctors* is deleted. This attribute was the join attribute with the relation *S2.Hospital*. In order to maintain the integrated schema without alterations the source *S2* is changed, adding the attribute *doctor* to relation *Hospitals* of *S2*. Obviously the transformation process is also modified changing the join attribute.

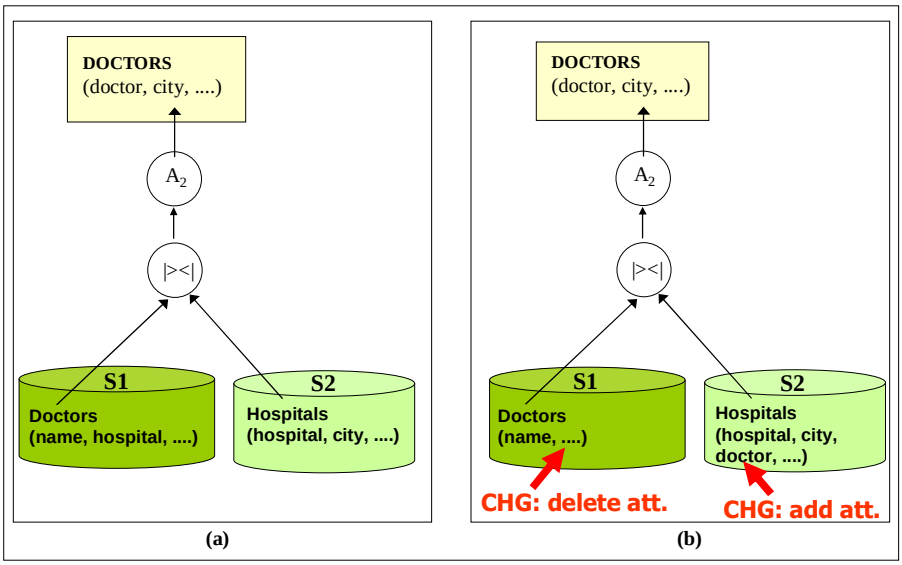


Figure 6.3: Source change propagation, where other source is modified. (a) Before change. (b) After change.

We now present the analogy of the situations of quality changes with the previous presented situations of source schema changes. In each case, we consider the three different types of changes we manage in this thesis: *source quality change*, *transformation graph change*, and *user quality requirement change.*

Situations:

(i) In source schema evolution, this is the case where changes propagate to the integrated schema. In the problem of quality changes, propagation occurs automatically. If the quality change is on a source or on the transformation graph, data targets quality automatically changes. If the change is on a user quality requirement, data targets quality does not change. In both cases the DIS probably goes to a state of non-satisfaction of user quality requirements.
Figure 6.4 shows an example. Values for the factor *accuracy* are shown. The value in source *S1* changes from *0.8* to *0.7*. After the propagation to the data target, the value changes from *0.7* to *0.6*.

(ii) This is the case where changes are absorbed by the data transformation process. This means that a modification is done to the transformation in order to compensate the occurred change. This modification may be on the transformation graph structure or on some property of an activity of this graph. In quality changes problem this case occurs as follows. If source quality or a quality requirement changes, the transformation is modified, compensating the change. If the change occurs on the transformation, some other modification is applied to it in order to compensate the change.
We show two different examples for this case. In Figure 6.5 we continue with the example of Figure 6.4, but the source *accuracy* change is compensated adding a cleaning task to the data transformations. In Figure 6.6 we show an example where *freshness* factor is considered. Freshness value changes in source *S2* from *19* to *21*, and this change is compensated decreasing the cost of activity *A3* from *3* to *2*, in order to continue satisfying the required freshness.

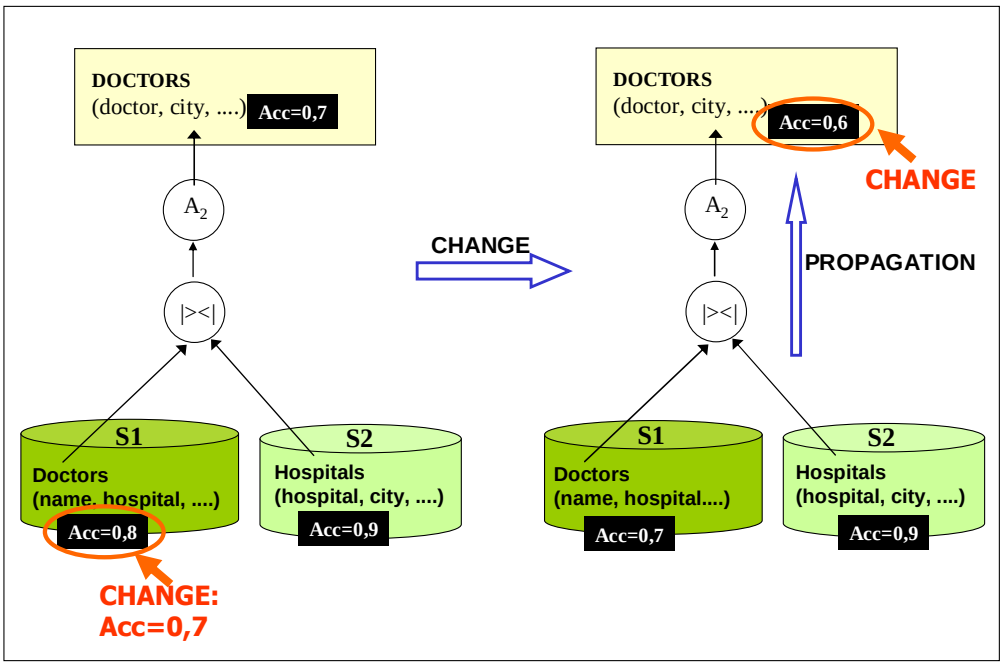


Figure 6.4: Example: source quality change propagation

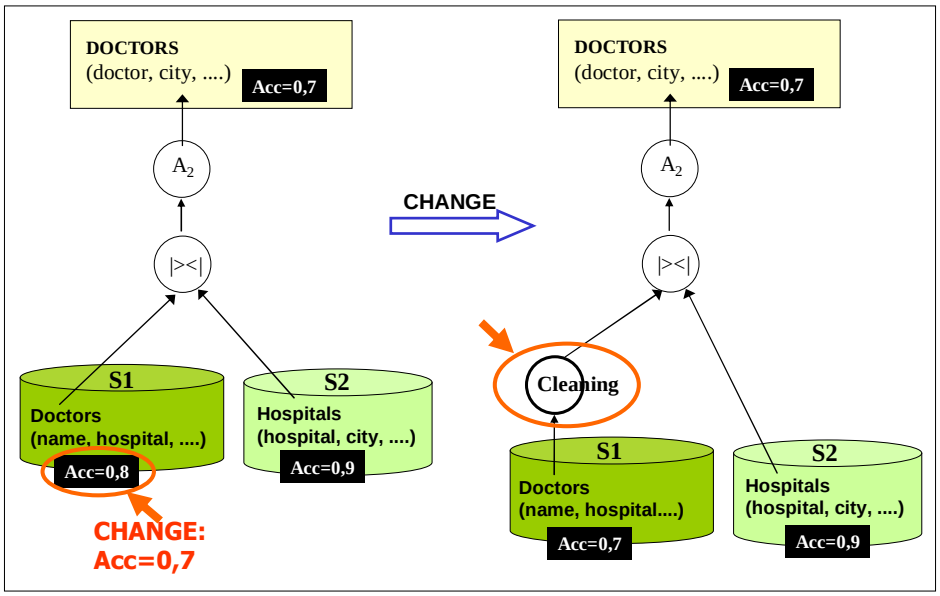


Figure 6.5: Example 1: source quality change absorbed by transformation

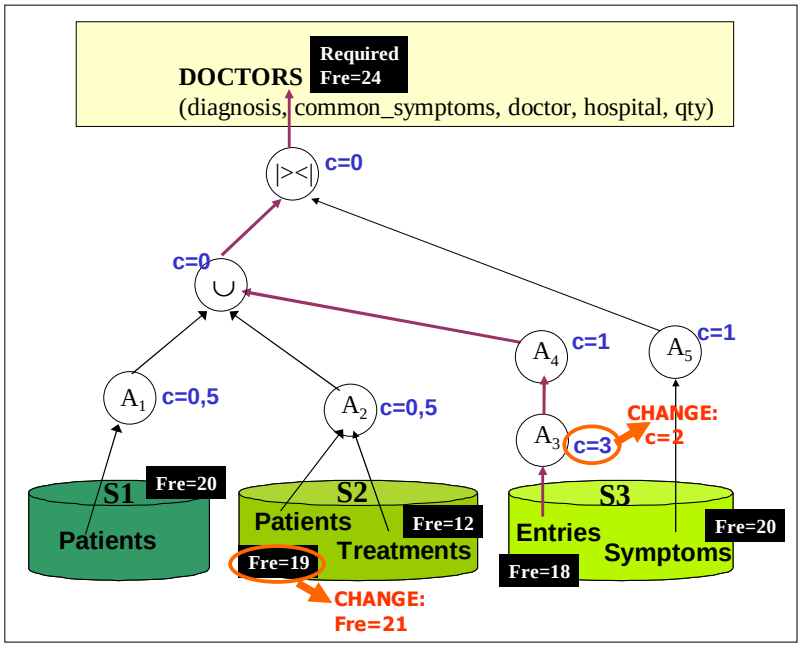


Figure 6.6: Example 2: source quality change absorbed by transformation

(iii) In this case, changes are compensated through a modification on a source. Considering the problem of quality changes, if the change occurs on the quality of a source, it is compensated changing the quality of another source. If the change occurs on a quality requirement or on the transformation process, a quality change is applied on a source in order to improve DIS quality. Figure 6.7 continues with example of Figure 6.5, but the change is compensated modifying quality of another source. In order to know which quality value is necessary in source *S2* for satisfying quality requirements, accepted configurations (defined in Chapter 4) are calculated.

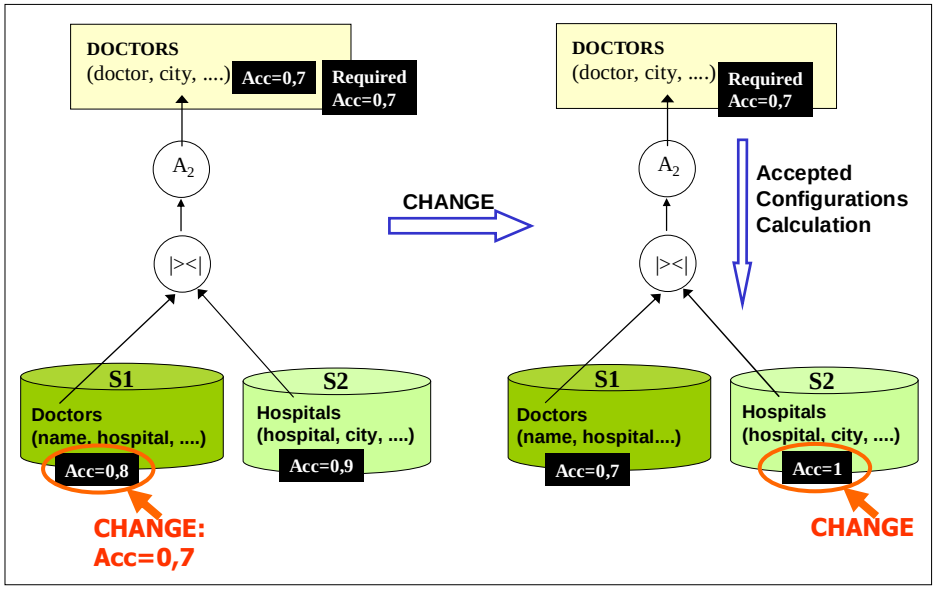


Figure 6.7: Example: source quality change compensated by change on another source

It is important to remark that these cases may occur or not, depending on the different particular systems and conditions. In the cases corresponding to (i) the DIS may result in the dissatisfaction of the required quality, while in the ones of (ii) and (iii), in general the actions are carried out in order to achieve requirements satisfaction again.

In Table 6.1 we summarize the situations presented above. Each situation can be seen as the consequence of the change.

Change \ Situation	**(i)**	**(ii)**	**(iii)**
Source quality	Data target quality changes	Transformation is modified	Another source changes its quality
User quality requirement	Data target quality does not change	Transformation is modified	Some source changes its quality
Transformation	Data target quality changes	Another change is applied to the transformation	Some source changes its quality

Table 6.1: Situations summary

2.2 Repairing Actions Classification

We classify repairing actions according to two different criteria. Therefore we have two orthogonal classifications.

The first one is directly obtained from the analysis of previous section. We classify repairing actions into two categories, according to what part of the DIS they modify:

- Actions on transformation graph.
  These are the actions that modify either the graph structure, either a property of certain graph activity.
- Actions on a source
  These are the actions that modify the quality or another characteristic of a source.

The second classification takes into account the kind of modification that the action applies. There are also two categories for the actions, according to this criterion:

- Actions that modify quality values or related properties
  These are the actions that are applied to quality values of the sources or to DIS properties that affect the quality values.
- Actions that modify DIS design
  These are the actions that change the design of the DIS, that is, the transformation structure, the transformation activities, the participant sources, etc.

Finally, a repairing action, for us, can belong to one of four different categories, shown in Table 6.2 as A, B, C or D.

Fst classif \ Snd classif	modifies quality values or related properties	modifies DIS design
on transformation graph	A	B
on a source	C	D

Table 6.2: Possible categories for a repairing action

"D" category corresponds to source schema modifications. In our work, we do not consider actions from this category, they fall outside of the scope.

For example, an action that modifies the cost of an activity of the quality graph, maintaining the semantics of the activity, (see example in Figure 6.6) is classified in "A" category. An action that adds an activity to the graph (see example in Figure 6.5) is classified in "B" category. An action that modifies the quality value of a source (see example in Figure 6.7) is classified in "C" category.

2.3 Quality Tuning

In order to determine the appropriate actions to apply in each case, it is necessary to know how each modification of each element of the DIS affects the resulting quality values. This behavior strongly depends on the quality factor we are considering. Therefore we do this analysis independently for each factor, freshness and accuracy. However, we also study how an action applied as a consequence of a change on one quality factor may affect the values of the other.

In the following we analyze the possible adjustments that can be made to the DIS in order to repair its quality, after a change that generated the dissatisfaction of quality requirements occurred.

It is important to note that the modifications we propose are intended to maintain as much as possible the topology of the graph, that is, the DIS design.

2.3.1 Freshness

To know how the different DIS elements affect data targets' freshness we must know how this freshness is calculated.

Depending on the application domain, the kind of transformation activities, and the nature of the data, data target freshness may be calculated differently. Generalizing, we consider that freshness can be calculated at each node of the quality graph, by what we call *propagation function*, according to two possible criteria: (i) choosing the freshness of one of the predecessors of the node, or (ii) combining all predecessor nodes' freshness. Basing on this, we distinguish two cases in our analysis:

Case 1 - Propagation function chooses one freshness value

In these cases the most relevant characteristic is the existence of a *critical path* in the quality graph, which we present in the following.

In order to fix ideas we consider the following propagation function, which is one of the possible ones in this case (also used in Chapter 4):

For any activity node A:

$Freshness(A) = Max\ (\ Freshness(A_1) + Delay(A_1,A), \ldots, Freshness(A_n) + Delay(A_n,A)\ ) + Cost(A)$, where $A_1, \ldots, A_n$ are A predecessors

The *Delay* is the time that passes between the end-time of one activity and the start-time of its successor. The *Cost* of an activity is its processing time interval.

We found that in this case, given a target node of the quality graph, there exists at least one path from one source to the target node that fixes the freshness of this node, which we call *critical path*. This means that the other paths to that target node are not affecting the final freshness. We define path and critical path in [Marotta+05], but for this work we use an alternative definition presented in [Peralta-06], which is presented as follows:

Definition 6.1 (from [Peralta-06]): A *data path* in a quality graph is a sequence of nodes of the graph, where each node is connected to its successor in the sequence by a data edge. We denote a data path, giving the sequence of nodes that compose it, comma separated and between square brackets, for example $[A_0,A_1,A_3,A_4]$. We also use suspension points for omitting intermediate nodes, for example $[A_0,\ldots A_4]$. □

Definition 6.2 (from [Peralta-06][1]): Given a data path in a quality graph $[A_0,A_1,\ldots A_p]$, starting at a source node A_0, the *path freshness* is the freshness value propagated along the path (ignoring other nodes of the graph), i.e. it is the sum of source data freshness of the source node, the processing costs of the nodes in the path and the inter-process delays between the nodes:

$$\text{PathFreshness}([A_0,\ldots A_p]) = \text{SourceFreshness}(A_0) + \sum_{x=0..p} \text{Cost}(A_x) + \sum_{x=1..p} \text{InterProcessDelay}(A_{x-1},A_x)$$ □

InterProcessDelay is the time that necessarily passes between two successive activities, without considering synchronization delays. That is to say it may exist a time interval between two activities generated by the way one activity passes data to the other, the frequency, etc.

Definition 6.3 (from [Peralta-06] [2]): Given an activity node A_p, a *critical path* for A_p is a data path $[A_0,\ldots A_p]$, from a source node A_0, for which the freshness of data produced by node A_p (delivered to each successor) equals the path freshness.

$$\text{Freshness}(A_p) = \text{PathFreshness}([A_0,\ldots A_p])$$

Given a target node T_i, a critical path for T_i is the critical path of its predecessor activity. □

The following is an example for showing a critical path in a quality graph.

Example 6.1

Retaking the example case managed in previous section, in Figure 6.8 we show a quality graph with the activities' costs and the processing delays attached to its nodes and edges respectively.

We first calculate the freshness of the target node *Doctors*:

$$\text{fr(Doctors)} = \text{fr}(A_7)$$

$$\text{fr}(A_7) = \text{Max}\ (\ \text{fr}(A_6) + \text{Delay}(A_6,A_7),\ \text{fr}(A_5) + \text{Delay}(A_5,A_7)\ ) + \text{Cost}(A_7)$$
$$\text{fr}(A_7) = \text{Max}\ (\ \text{fr}(A_6),\ \text{fr}(A_5)\ )$$

[1] With minimum nomenclature modifications.

[2] With minimum nomenclature modifications.

$fr(A_6)$ = Max ($fr(A_1)$ + Delay(A_1,A_6), $fr(A_2)$ + Delay(A_2,A_6), $fr(A_4)$ + Delay(A_4,A_6)) + Cost(A_6)
$fr(A_6)$ = Max ($fr(A_1)$ + 0.5, $fr(A_2)$, $fr(A_4)$)

$fr(A_5)$ = fr(Symptoms) + Delay(Symptoms,A_5) + Cost(A_5)
$fr(A_5)$ = 20 + 0 + 1 = 21

$fr(A_1)$ = fr(S1.Patients) + Delay(S1.Patients,A_1) + Cost(A_1)
$fr(A_1)$ = 20 + 0 + 0.5 = 20.5

$fr(A_2)$ = Max (fr(S2.Patients) + Delay(S2.Patients, A_2), fr(Treatments) + Delay(Treatments,A_2)) + Cost(A_2)
$fr(A_2)$ = Max (19, 12) + 0.5 = 19.5

$fr(A_4)$ = $fr(A_3)$ + Delay(A_3,A_4) + Cost(A_4)
$fr(A_4)$ = $fr(A_3)$ + 2 + 1

$fr(A_3)$ = fr(Entries) + Delay(Entries,A_3) + Cost(A_3)
$fr(A_3)$ = 16 + 3 = 19

Substituting:

$fr(A_4)$ = 19 + 2 + 1 = 22

$fr(A_6)$ = Max (20.5 + 0.5, 19.5, 22) = 22

$fr(A_7)$ = Max (22, 21) = 22

fr(Doctors) = 22

The following are the paths of the quality graph that start in a source node and end in the target node:

[S1.Patients, A_1, A_6, A_7, Doctors]
[S2.Patients, A_2, A_6, A_7, Doctors]
[S2.Treatments, A_2, A_6, A_7, Doctors]
[S3.Entries, A_3, A_4, A_6, A_7, Doctors]
[S3.Symptoms, A_5, A_7, Doctors]

We calculate the freshness values of the paths, applying the formula *PathFreshness($[A_0,\ldots A_p]$) =*
SourceFreshness(A_0) + $\sum_{x=0..p}$ Cost(A_x) + $\sum_{x=1..p}$ InterProcessDelay(A_{x-1},A_x).

PathFreshness([S1.Patients, A_1, A_6, A_7, Doctors]) = 20 + 0.5 + 0.5 = 21

Analogously we calculate the other ones, in particular, the following:

PathFreshness([S3.Entries, A_3, A_4, A_6, A_7, Doctors]) = 16 + 4 + 2 = 22

We see that:

fr(Doctors) = PathFreshness([S3.Entries, A_3, A_4, A_6, A_7, Doctors])

According to Definition 6.3, [S3.Entries, A_3, A_4, A_6, A_7, Doctors] is the critical path for Doctors.

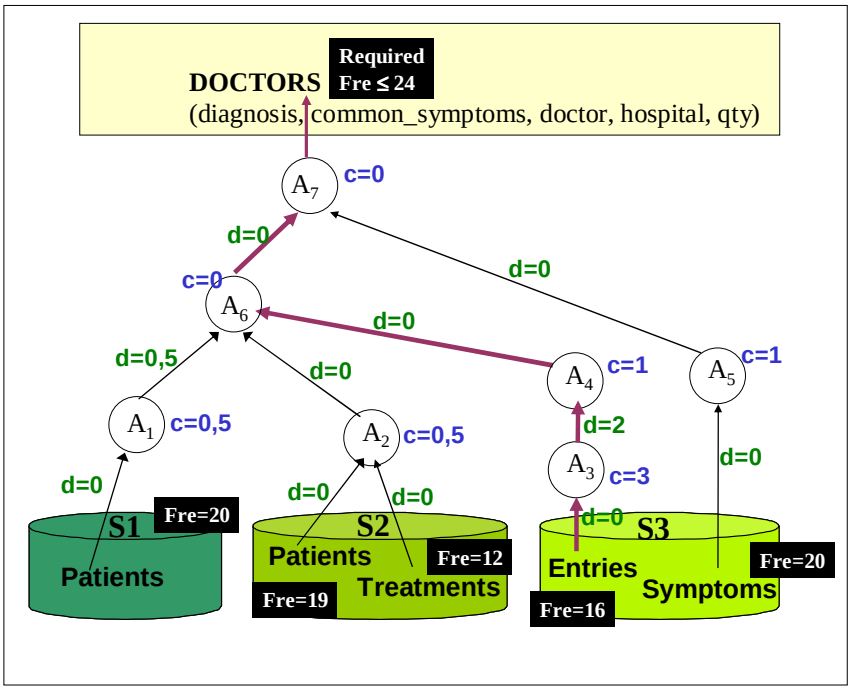


Figure 6.8: Critical Path

◊

The set of critical paths may change at any moment; if a path *p1* is a critical path and its freshness decreases, perhaps some other path *p2* becomes a critical path. On the other hand, if a path *p1* is a critical path and some other path *p3* increases its freshness, *p3* may become a critical path, while *p1* may not be a critical path any more.

From the study of critical paths we deduce that to decrease target freshness we must decrease the freshness of these paths.

In the following we analyze the modifications we can apply in order to repair the DIS after a change that generated the dissatisfaction of the quality requirements, considering quality requirements without probabilities.

If the occurred change was on a source quality or on the transformation graph, the critical path may have changed, i.e. is not the same path as before. In the case of source quality change, this happens if the changed source node did not belong to the critical path before the change. In this case the critical path becomes the one that goes from the changed source node to the target node. This is because the changed source node is the one that caused the dissatisfaction of quality requirements. This assertion can be clearly shown through an example. See Example 6.2.

Example 6.2

Consider the quality graph of Example 6.1, see Figure 6.8. The critical path is [S3.Entries, A_3, A_4, A_6, A_7, Doctors]. Suppose source relation *S2.Patients* changes its freshness from *19* to *24*. This generates the dissatisfaction of the requirement: *freshness* ≤ 24, since, now the freshness of *Doctors* is equal to *24.5* (calculating this freshness as it was calculated in Example 6.1).

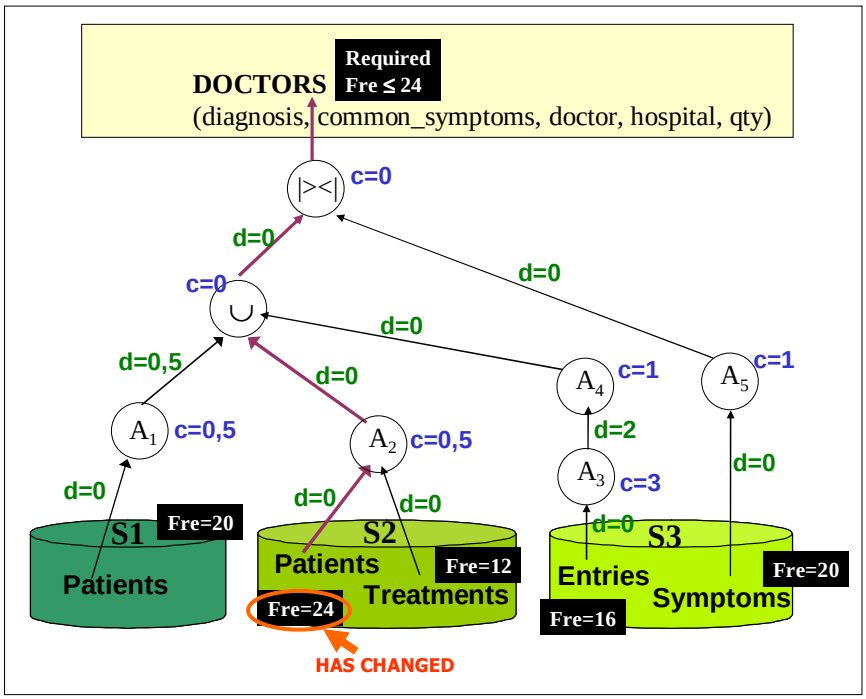


Figure 6.9: Source quality change - Critical Path change

In this case we have as change, the source quality change of *S2.Patients*. This source did not belong to the critical path before the change. We calculate now the critical path, applying its definition. It is the path whose freshness is equal to the freshness of the data target, which is *Doctors*.

PathFreshness ([S2.Patients, A_2, A_6, A_7, Doctors]) = 24.5

fr (Doctors) = 24.5

Therefore, the critical path now is: [S2.Patients, A_2, A_6, A_7, Doctors]. We can see this comparing Figure 6.8 and 6.9.

◊

Therefore, to repair DIS quality:

1) In the cases of source quality change, where the source node did not belong to the critical path, and transformation graph change, we must re-calculate the critical path.
2) We must decrease the freshness value of the critical path. The modifications that may be useful to apply are:
    a. decreasing costs of activities of the critical path
    b. decreasing inter-process delays of the critical path
    c. eliminating an activity of the critical path
    d. decreasing freshness value of the source that belongs to the critical path
    e. eliminating a source that belongs to the critical path (specially when the change was on this source)

f. substituting a source that belongs to the critical path, by another one (specially when the change was on this source)

3) After decreasing the value of critical path freshness, it may happen that other path becomes the critical path. If the quality requirements are still not being satisfied, we must decrease freshness value of the new critical path. The same repeats successively.

Case 2 - Propagation function combines freshness values

These are the cases where the propagation function associated to an activity of the transformation graph, calculates a freshness value, for the resulting data, from the combination of the freshness values of the input data. For instance, a propagation function for an activity that applies a join operation between two input relations, may calculate freshness of the resulting data as the average of the input relations' freshness.

In this case there is not a critical path, since we can influence on the target freshness modifying several different paths. This is because the freshness resulting from each activity may be modified changing any of the input data freshness.

For the previously exposed reasons, there are more possible modifications to repair DIS quality in this case than in Case 1. That is to say, in Case 1, to improve target freshness it only has sense to "touch" the critical path. However, in this case the set of DIS components we may "touch" to improve target freshness after a change that generated quality requirements dissatisfaction, is wider. The DIS components we may modify achieving an improvement are the following:

1) Any source that is connected to the target node in the quality graph

2) Any part of the transformation graph that is connected to the target node in the quality graph

If the occurred change was on the transformation graph or on the user quality requirements, the accepted configurations must be re-calculated.

If we opt to modify 1), we propose the following procedure: Choose a restriction vector from the accepted configurations of the target node, and try to make the sources satisfy these restrictions. For choosing the most convenient restriction vector, useful criteria would be to choose the vector which has the greatest amount of sources satisfying the restrictions, or to take into account which sources offer more possibilities of changing their quality.

If we opt to modify 2), the modifications that may decrease the freshness of the target are:

a. decreasing costs of activities

b. decreasing inter-process delays

c. eliminating an activity

d. substituting a source by another one

e. eliminating a source

Case 1 and Case 2

In both cases, if the user quality requirement/s considered is of type "probability" and the generated dissatisfaction is that the DIS Quality Certainty does not satisfy this requirement, then the modifications suggested in both cases have as secondary effect that the probabilities of the sources satisfaction of quality requirements increase, and then certainty improves.

2.3.2 Accuracy

The case of accuracy factor is very similar to Case 2 of previous section, since we consider that for accuracy, in general, the propagation function takes a combination of the accuracy values of the input relations. For example, a typical and very simple propagation function is the one proposed in [Naumann+99], which lays on the assumption that errors are uniformly distributed in the input relations. In this proposal, for the join operation, the accuracy of the joined data is calculated as the product of the accuracies of both input relations.

Due to the characteristic of the propagation functions of combining accuracies of the input relations to obtain the resulting accuracy, like in Case 2 of freshness, all the input accuracy values influence the resulting accuracy. Therefore, the accuracy resulting from each activity may be modified changing any of the input data accuracies.

When a change that generated quality requirements dissatisfaction has occurred, like in the case of freshness, the DIS components we may modify to repair DIS quality, are the following:

1) Any source that is connected to the target node in the quality graph
2) Any part of the transformation graph that is connected to the target node in the quality graph

In addition, the statements about re-calculation of accepted configurations and the procedure to follow if we want to modify 1) presented for freshness, also applies to accuracy.

In the case we opt to modify 2), the modifications that may increase accuracy of the target are:

a. increasing effectiveness of cleaning activities (augmenting the percentage of information that is corrected).
b. adding cleaning activities
c. substituting a source by another one
d. eliminating a source

2.3.3 Freshness vs. Accuracy

Freshness and accuracy are not totally independent factors.

It is true that a change on one factor does not automatically generate a change on the other one. For example, if a certain source increases or decreases its accuracy, without affecting DIS structure or functioning, freshness of the source or of the DIS is not affected at all. However, when the occurred change affects DIS quality and, as a consequence, some DIS characteristics are modified, the other factor may be affected. In summary, when we modify the system in order to improve the values of one factor, the values of the other factor may get worse.

We can cite concrete examples of this fact, basing on the possible modifications suggested in previous sections:

- Eliminating an activity of the quality graph to improve target node freshness.
  If the eliminated activity is a cleaning activity perhaps target node accuracy decreases.
- Substituting a source by another one whose data has a lower freshness value, to improve target node freshness.
  If accuracy of the new source data is lower than accuracy of the old source, target node accuracy decreases.

- Substituting a source by another one whose data has a better accuracy value, to improve target node accuracy.
  If freshness of the new source data is worse than freshness of the old source, target node freshness value increases.
- Augmenting effectiveness of cleaning activities to improve target node accuracy.
  This modification probably augments the cost of the activities, and in that case it probably increases target node freshness value.
- Adding cleaning activities.
  This modification may increase the freshness value of the target node.

Example 6.3

Retaking example of Figure 6.5, we now consider also freshness factor. Before the change on the accuracy of source *S1*, freshness of the data target *Doctors* was equal to *18*. After the source change, an activity is added to the graph maintaining the accuracy of target *Doctors*, but this action has as secondary effect that freshness changes to *22*. Figure 6.10 illustrates this example.

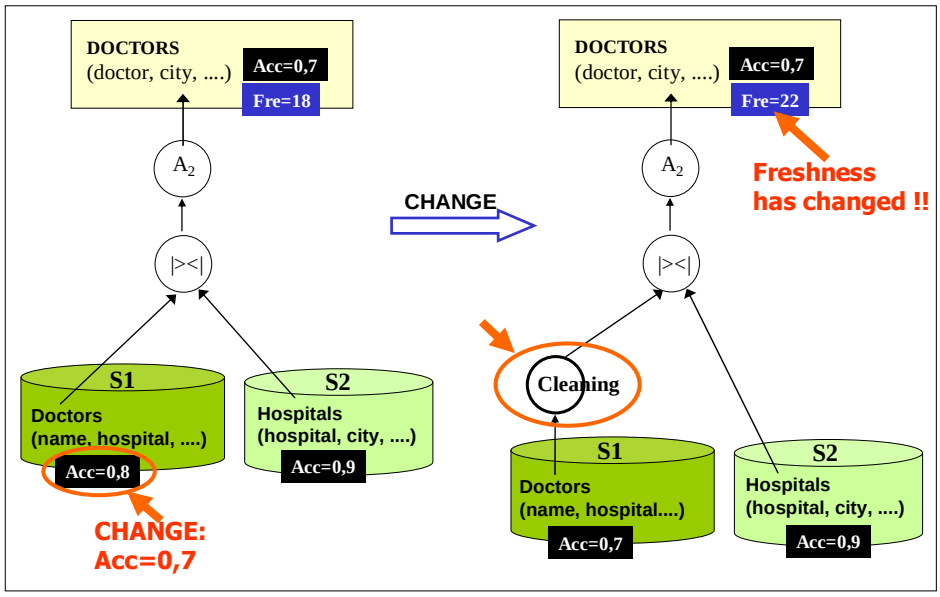


Figure 6.10: Accuracy change indirectly causes freshness change

◊

3. Quality Repair in the QMS

The QMS receives two kinds of events, which communicate that DIS quality has suffered a change and now is not satisfying user quality requirements. These two kinds of events are: *QValuesDissatisfaction* and *ProbabilityDissatisfaction*. The first means that there was a change that caused the dissatisfaction of quality requirements that are not "probability" requirements. The second means that there was a change that caused the dissatisfaction of "probability" requirements,

that is, DIS quality certainty does not reach all the probability values given by the users in the requirements.

The received events provide information about the occurred change, such as the type of change, the DIS component where it occurred, etc. On the other hand, more useful information can be found in the meta-information database, called *estimations & statistics* (presented in Chapter 3). The QMS uses all this information for diagnosing the situation of the DIS and then determining the recommendable actions that are adequate for the obtained diagnostic. The concept of DIS situation involves not only its current state, but also its history and its probable future. The actions that are recommended are basically the ones presented and analyzed in Section 2.

In this section we address the problem of determining actions to recommend for repairing DIS quality, given a received event and statistical information. We divide this problem into two sub-problems: (1) arriving to an interpretation of DIS situation, and (2) from DIS situation, obtaining a list of recommended actions for repairing DIS quality.

Our intention is to provide a mechanism for solving this problem, which can be easily instantiated with different statistics, interpretations and rules. The mechanism is based on two sets of rules, one for deducing the interpretation, called *Interpretation Rules*, and the other for obtaining the actions, called *Repairing Rules*. Figure 6.11 shows the proposed processes for DIS Quality Repair. We propose some of these rules, as examples of which may be done. The DIS administrator may define new rules, interpretations, and actions. It is also possible to define new statistics, but in that case it is necessary to implement the way these statistics will be maintained.

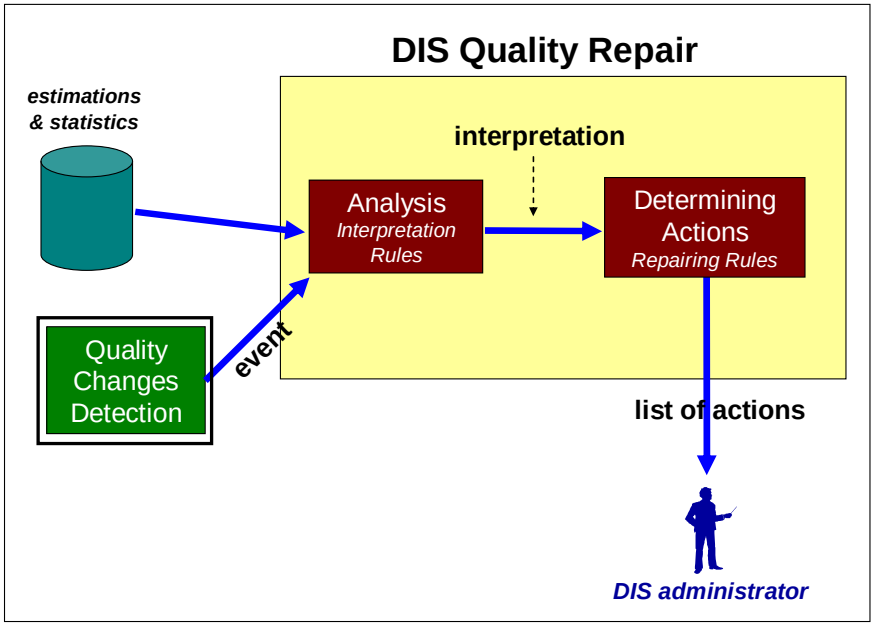


Figure 6.11: DIS Quality Repair Architecture

3.1 Analysis of DIS Situation

For the tasks of analyzing and interpreting the DIS situation we define *Interpretation Rules*. These rules give an *interpretation* of the main problem the DIS is having, given a change *event*, a *condition* satisfied by the change, and some *statistical information*.

The *event* is received from the Change Detection module and, as said before, it can be a *QValuesDissatisfaction* event or a *ProbabilityDissatisfaction* event. It notifies that a change that generated quality requirements dissatisfaction has occurred, and it provides information about the occurred change. Table 6.3 shows the structure of the possible events.

The *condition* of the rules is a condition over the event's attributes.

Event Class	Attribute	Attribute Description
QValuesDissatisfaction	QGraph	Quality Graph where the change occurred.
	Requirements	Set of requirements that are not satisfied.
	OriginalChange	Change that generated the level-1 event.
	OriginalSource	Source where the change occurred.
	GraphChange	Kind of change that occurred on the transformation graph.
	ChangedActivity	Graph activity where a change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the original change.
ProbabilityDissatisfaction	QGraph	Quality Graph where the change occurred.
	QFactor	Quality factor.
	OriginalChange	Change that generated the level-1 event.
	OriginalSource	Source where the change occurred.
	GraphChange	Kind of change that occurred on the transformation graph.
	ChangedActivity	Graph activity where a change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the original change.

Table 6.3: Events structure

Statistical information is extracted from the *estimations & statistics* part of the meta-information managed by the QMS, which contains current information and estimations about DIS properties and quality models, quality measurements, and also historical information about them. We propose to use functions that query and process this information returning a boolean answer.

The *interpretation* is how we interpret the DIS situation, after observing the change event and querying the statistical information. It supposes what has happened in a source, in an activity, etc.

As expressed before, we do not intend to provide an exhaustive or complete set of rules or interpretations, the idea is to give the general mechanism that makes possible the treatment of as many cases as wanted.

The following are some possible **interpretations** for DIS situations:

- The kinds of accuracy errors coming from sources have changed.
  Data present errors that are different from the ones that it used to present before. The cleaning activity assigned to these data is not working as effectively as before.

- Inconvenient source.
  A certain source changes very frequently, causing quality requirements dissatisfaction each time.
- Source punctual change.
  A certain source had a quality change causing quality requirements dissatisfaction, but this was an isolated event.[1]
- There is a problem with data accessibility in a source.
  Query efficiency has decreased in the source, affecting DIS freshness.
- There is a problem with the volume of data coming from certain source.
  The volume of data coming from a source has increased and affects DIS freshness.
- User quality requirement is constantly changing.
  In certain data target, user quality requirement for a quality factor is changing very frequently.

We specify the interpretations as object classes, which are instantiated in each particular case. Table 6.4 shows these classes.

Interpretation Class	Attribute	Attribute Description
InconvenientSource	Source	Referred source.
	QFactor	Quality Factor for which the source is inconvenient.
	Requirements	Requirements that are not satisfied.
SourcePunctualChange	Source	Referred source.
	QFactor	Quality Factor that has changed.
	Requirements	Requirements that are not satisfied.
SourceError-typesChange	Activity	Cleaning activity that processes the data.
SourceDataAccessibilityProblem	Activity	Extraction activity that extracts the data.
SourceDataVolumeProblem	Activity	Extraction activity that extracts the data.
ConstantlyChangingUserQualityRequirement	Requirement	Referred requirement.

Table 6.4: Interpretations structure

In the following we present the **statistics** used in our rules. The information comes from *estimations & statistics* meta-information. In some cases we define a function that obtains it, and in other cases

[1] Note that the occurred change was a change on the source quality model. We call it a punctual change because the model does not change very frequently.

we use the information contained in the change event, which also comes from the same meta-information database.

- Data_Volume_Increased
  Given an activity, it returns TRUE if the volume of data that is processed by the activity has increased, according to the statistics, and FALSE otherwise.
- Frequently_Changes
  It may receive as input a source or a quality requirement, and a quality factor. It returns TRUE if the source or requirement has changed its values for the quality factor, with certain frequency and recently. Otherwise it returns FALSE. The function implementation determines the frequency and time interval considered, according to the application domain and needs.
- Activity cost has increased
  The attribute *GraphChange* of the change event (*QValuesDissatisfaction* or *ProbabilityDissatisfaction*) has the kind of change the transformation graph has suffered. When this value is "Activity-cost", it means that the cost of an activity has increased, affecting target freshness.
- Activity effectiveness has decreased
  The attribute *GraphChange* of the change event (*QValuesDissatisfaction* or *ProbabilityDissatisfaction*) has the kind of change the transformation graph has suffered. When this value is "Activity-effectiveness", it means that the percentage of cleaned data by certain cleaning activity has decreased, affecting target accuracy.

In the following we sketch some possible **rules**.

Note: In the rules we specify interpretations as functions that create the interpretation with the attribute values they receive as input.

Interpretation Rules

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = "QModelChange"
STATISTIC:	Frequently_Changes (e.OriginalSource, GetQFactor(e.Requirements))
INTERPRETATION:	Inconvenient_Source (e.OriginalSource, GetQFactor(e.Requirements), e.Requirements)

EVENT:	e: *ProbabilityDissatisfaction*
CONDITION:	e.OriginalChange = "QModelChange"
STATISTIC:	Frequently_Changes (e.OriginalSource, GetQFactor(e.Requirements))
INTERPRETATION:	Inconvenient_Source (e.OriginalSource, GetQFactor(e.Requirements), e.Requirements)

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = "QModelChange"
STATISTIC:	NOT Frequently_Changes (e.OriginalSource, GetQFactor(e.Requirements))
INTERPRETATION:	Source_Punctual_Change (e.OriginalSource, GetQFactor(e.Requirements), e.Requirements)

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = “TGraphChange”
STATISTIC:	e.GraphChange = “activity-effectiveness”
INTERPRETATION:	Source_Error-types_Change (e.ChangedActivity)

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = “TGraphChange”
	AND GetQFactor(e.Requirements) = “freshness”
	AND ExtractionActivity (e.ChangedActivity)
STATISTIC:	e.GraphChange = “activity-cost” AND
	NOT Data_Volume_Increased (e.ChangedActivity)
INTERPRETATION:	Source_Data_Accessibility_Problem (e.ChangedActivity)

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = “TGraphChange”
	AND GetQFactor(e.Requirements) = “freshness”
	AND ExtractionActivity (e.ChangedActivity)
STATISTIC:	e.GraphChange = “activity-cost”
	AND Data_Volume_Increased (e.ChangedActivity)
INTERPRETATION:	Source_Data_Volume_Problem (e.ChangedActivity)

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = “QReqChange”
STATISTIC:	Frequently_Changes (e.ChangedReq)
INTERPRETATION:	Constantly_Changing_User_Quality_Requirement (e.ChangedReq)

Used auxiliary functions:

Function *ExtractionActivity* receives an activity node of the quality graph, and returns a Boolean that indicates if it is an extraction activity or not.

Function *GetQFactor* returns the quality factor of the user quality requirements.

3.2 Determination of Recommended Actions

Once the situation of the DIS was diagnosed, generating an interpretation for it, the QMS determines a ranked-list of actions that may be applied to the DIS to repair its quality. Then it provides this list to the DIS administrator, who decides what to do, choosing one of the options given by the system. The system will automatically update its models and all necessary meta-information for continuing working.

In Section 2 we analyzed many different actions that may be carried out in order to recover DIS quality. In this section we consider those actions as a set from which the recommended ones are picked, depending on the current interpretation.

The QMS selects the most adequate actions for each interpretation. We specify this selection through a set of rules of the form *interpretation, condition → action-list*, which we call *Repairing Rules*.

The intention in giving this ranked-list of actions to the user (DIS administrator) is to provide him some clue, some line, or simply, useful information, for arriving to a good solution for the DIS quality problem. The idea is that the user chooses one of the recommended actions and uses other functionalities of the QMS for obtaining more information, which may be needed to apply the action.

We present here some repairing rules that apply to the interpretations presented in previous section, and as in the case of interpretation rules, they are not meant to be exhaustive. Our intention is to show some examples of these rules.

We first present an intuitive description of some of the given rules:

RR1 -
The QMS detected an inconvenient source for freshness. This means that the source is really compromising the maintenance of DIS freshness. The most recommended action is to eliminate this source. As a second possibility it is recommended to substitute the source by another one. The following recommended actions are to decrease some activity cost, to decrease inter-process delays or to eliminate some activity, all of them from the critical paths of the affected requirements.

RR2 -
The QMS detected an inconvenient source for accuracy. This means that the source is really compromising the maintenance of DIS accuracy. The most recommended action is to eliminate this source. As a second possibility it is recommended to substitute the source by another one. The following recommended actions are to increase the effectiveness of some cleaning activities or to add a new cleaning activity, considering activities that affect the requirements that are not being satisfied. The last recommended action is to modify the accuracy of another source, which supposes a negotiation with it. For this, the user may obtain the *accepted configurations* for the affected requirements in order to know which are the changes he needs.

RR3 -
The QMS detected that there was a source punctual change of freshness. The most recommended action is to decrease the costs of activities that belong to the critical paths of the affected requirements. The following recommended actions are to decrease inter-process delays and to eliminate activities, from the same paths. As a final option, it is recommended to do nothing and wait for taking a decision.

RR5 -
The QMS detected that the types of errors that arrive to certain cleaning activity, have changed. The cleaning techniques that are applied to these source-data are not as effective as before. The most recommended action is to substitute the cleaning activity by another one that works better with the current types of errors. Secondly it is suggested to add a new cleaning activity to achieve the necessary accuracy.

RR6 -
The QMS detected that data from certain source is not being extracted as efficiently as before. The first proposed action is to substitute the extraction activity by another one that is capable to obtain these data more efficiently. The second proposed action is to achieve an improvement in source data accessibility, that is to say, to ask the source for an improvement on its access paths.

We specify the repairing actions as object classes, which are instantiated in each particular case. The information contained in their attributes is the information that the QMS provides the user together with each action, as complementary information. For example, in the case of "EliminatingSource", the complementary information is the source he should eliminate. In the case of "DecreasingActivityCost", the complementary information are the paths where the activities that may be modified belong. Table 6.5 shows the object classes.

Action Class	Attribute	Attribute Description
EliminatingSource	Source	Source to eliminate.
SubstitutingSource	Source	Source to substitute.
DecreasingActivityCost	CPaths	Set of critical paths.
DecreasingInter-processDelays	CPaths	Set of critical paths.
EliminatingActivity	CPaths	Set of critical paths.
IncreasingActivityEffectiveness	CleaningActivities	Set of cleaning activities.
AddingCleaningActivity	Requirements	Set of requirements to be satisfied.
ModifyingOtherSource	Source	Source that has changed.
	Requirements	Set of requirements to be satisfied.
Waiting	—	—
SubstitutingActivity	Activity	Activity to substitute.
ModifyingSourceAccessibility	Activity	Extraction activity.
NotifyUser	Requirement	Requirement that is constantly changing.
SubstitutingSomeSource	CPaths	Set of critical paths.

Table 6.5: Actions structure

As said at the beginning of this section, once the user chooses one of the actions proposed by the QMS, the QMS must automatically update its models and all necessary meta-information for continuing working. In [Peralta-06], the author specifies a set of actions, called *improvement actions*, which are the possible basic modifications to the quality graph. They are for example, *add_node*, *add_edge*, *remove_node*, *add_property*, etc. In addition, combinations of these elemental actions, macro actions, are defined. The repairing actions we propose can be easily written in terms of these improvement actions, adding some input information that should be obtained through the interaction with the user. This allows having the specification of the correction to the quality graph after the user chooses a repairing action. For example, repairing action *SubstitutingSource*, with source *S*, should be transformed to improvement action *replaceNode*, with the following arguments provided by the user: the quality graph, the node by which the source node *S* is replaced and the properties of this node.

Note: In the same way as for interpretations, in the rules we specify actions as functions that create the action with the attribute values they receive as input.

Repairing Rules:

RR1 -
INTERPRETATION: I: *InconvenientSource*
CONDITION: I.QFactor = "freshness"
ACTION-LIST: 1- EliminatingSource (I.Source)
2- SubstitutingSource (I.Source)
3- DecreasingActivityCost (CriticalPaths (I.Requirements))
4- DecreasingInter-processDelays (CriticalPaths (I.Requirements))
5- EliminatingActivity (CriticalPaths (I.Requirements))

RR2 -
INTERPRETATION: I: *InconvenientSource*
CONDITION: I.QFactor = "accuracy"
ACTION-LIST: 1- EliminatingSource (I.Source)
2- SubstitutingSource (I.Source)
3- IncreasingActivityEffectiveness (CleaningActivities (I.Requirements))
4- AddingCleaningActivity (I.Requirements)
5- ModifyingOtherSource (I.Source, I.Requirements)

RR3 -
INTERPRETATION: I: *Source_Punctual_Change*
CONDITION: I.QFactor = "freshness"
ACTION-LIST: 1- DecreasingActivityCost (CriticalPaths (I.Requirements))
2- DecreasingInter-processDelays (CriticalPaths (I.Requirements))
3- EliminatingActivity (CriticalPaths (I.Requirements))
4- Waiting

RR4 -
INTERPRETATION: I: *Source_Punctual_Change*
CONDITION: I.QFactor = "accuracy"
ACTION-LIST: 1- IncreasingActivityEffectiveness (CleaningActivities (I.Requirements))
2- AddingCleaningActivity (I.Requirements)
3- ModifyingOtherSource (I.Source, I.Requirements)
4- Waiting

RR5 -
INTERPRETATION: I: *Source_Error-types_Change*
CONDITION: TRUE
ACTION-LIST: 1- SubstitutingActivity (I.Activity)
2- AddingCleaningActivity (I.Requirements)

RR6 -
INTERPRETATION: I: *Source_Data_Accessibility_Problem*
CONDITION: TRUE
ACTION-LIST: 1- SubstitutingActivity (I.Activity)
2- ModifyingSourceAccessibility (I.Activity)

RR7 -
INTERPRETATION: I: *Source_Data_Volume_Problem*
CONDITION: TRUE
ACTION-LIST: 1- SubstitutingActivity (I.Activity)

RR8 -
INTERPRETATION: I: *Constantly_Changing_User_Quality_Requirement*
CONDITION: GetQFactor(I.Requirement) = "freshness"
ACTION-LIST: 1- NotifyUser (I.Requirement)
2- DecreasingActivityCost (CriticalPaths (I.Requirement))
3- DecreasingInter-processDelays (CriticalPaths (I.Requirement))
4- SubstitutingSomeSource (CriticalPaths (I.Requirement))

Used auxiliary functions:

Function *CriticalPaths* receives a set of requirements and returns a set of paths that are the critical paths of the received requirements.

Function *CleaningActivities* receives a set of requirements and returns all the cleaning activities that process data that is involved by the requirements.

Note: In rules *RR1*, *RR3* and *RR8*, we assume the case where freshness propagation function chooses one of the input freshness values for giving the resulting freshness, when applied to an activity (Case 1 of Section 2.3.1 of this Chapter).

4. Example

In Chapter 5, Section 6 we had presented an example where some relevant quality changes were detected. In this section we continue with the same example, assuming the QMS Quality Repair module has received the event generated by the Change Detection module.

4.1 First Change

The cost of activity A2 has changed. Figure 6.12 shows the quality graph and its change.

The received event was:

QValuesDissatisfaction, with attributes:

QGraph = "MeteoGraph"
Requirements = { <T1, Rmax1>, <T2, Rmax2>, <T3, Rmax3> }
OriginalChange = "TGraphChange"
OriginalSource = NULL
GraphChange = "activity-cost"
ChangedActivity = A2
ChangedReq = NULL
Timestamp = 5/11-3

Remember that the requirements were the following:

ρ_V (T1) (UserQualityRequirement) (Rmax1) = < freshness, maximum, 72, NULL >
ρ_V (T2) (UserQualityRequirement) (Rmax2) = < freshness, maximum, 48, NULL >
ρ_V (T3) (UserQualityRequirement) (Rmax3) = < freshness, maximum, 2, NULL >

The following Interpretation Rule is applied:

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = "TGraphChange" AND GetQFactor(e.Requirements) = "freshness" AND ExtractionActivity (e.ChangedActivity)
STATISTIC:	e.GraphChange = "activity-cost" AND NOT Data_Volume_Increased (e.ChangedActivity)
INTERPRETATION:	Source_Data_Accessibility_Problem (e.ChangedActivity)

The condition is satisfied, since: ExtractionActivity (A2) = TRUE

Statistic is satisfied because: Data_Volume_Increased (A2) = FALSE

Interpretation **Source_Data_Accessibility_Problem** is generated, with attribute Activity = A2.

The following Repairing Rule is applied:

INTERPRETATION:	I: *Source_Data_Accessibility_Problem*
CONDITION:	TRUE
ACTION-LIST:	1- SubstitutingActivity (I.Activity) 2- ModifyingSourceAccessibility (I.Activity)

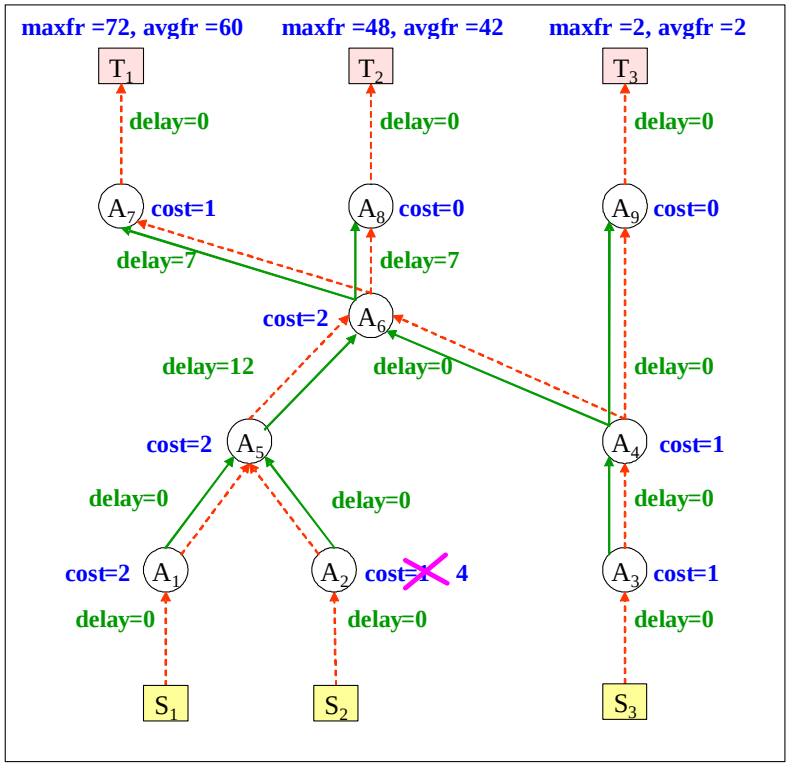


Figure 6.12: Transformation graph change

The QMS proposes the system administrator, in first place, to substitute the activity A2 by another one that extracts source S2 data in a more efficiently way. In second place it proposes him to ask

source S2 administrator to improve data access, taking into account that this accessibility has decreased.

4.2 Second Change

Two indicators (maximum and expectation) have changed in the quality model of source S1.

The received event was:

QValuesDissatisfaction, with attributes:

QGraph = "MeteoGraph"
Requirements = { <T1, Rmax1>, <T2, Rmax2> }
OriginalChange = "QModelChange"
OriginalSource = S1
GraphChange = NULL
ActivityChange = NULL
ChangedReq = NULL
Timestamp = 5/11-3

The following Interpretation Rule is applied:

EVENT:	e: *QValuesDissatisfaction*
CONDITION:	e.OriginalChange = "QModelChange"
STATISTIC:	NOT Frequently_Changes (e.OriginalSource, GetQFactor(e.Requirements))
INTERPRETATION:	Source_Punctual_Change (e.OriginalSource, GetQFactor(e.Requirements), e.Requirements)

Statistic is satisfied because: Frequently_Changes (S1, freshness) = FALSE

Interpretation **Source_Punctual_Change** is generated, with attributes:
Source = S1
QFactor = freshness
Requirements = { <T1, Rmax1>, <T2, Rmax2> }

The following Repairing Rule is applied:

INTERPRETATION:	I: *Source_Punctual_Change*
CONDITION:	I.QFactor = "freshness"
ACTION-LIST:	1- DecreasingActivityCost (CriticalPaths (I.Requirements))
	2- DecreasingInter-processDelays (CriticalPaths (I.Requirements))
	3- EliminatingActivity (CriticalPaths (I.Requirements))
	4- Waiting

The critical paths in this case are the ones that start at S1, since the value of S1 has caused the dissatisfaction of the target requirements. Therefore, the critical paths are the following (see Figure 6.12):

[S1, A1, A5, A6, A7, T1]

[S1, A1, A5, A6, A8, T2]

Therefore, the QMS suggests the administrator to decrease the cost of activities of these paths. For example, decreasing the cost of A1, A5 or A6, helps to the satisfaction of both requirements Rmax1 and Rmax2. The following suggested actions are to decrease inter-process delays and to eliminate activities from the same paths. As a final option it suggests to do nothing and wait for the source to return to the previous values itself.

5. Summary

This chapter presents, on one hand, an analysis of the possible modifications to the DIS for recovering from a quality change, and on the other hand, the mechanism that the QMS applies for automatically determining the most suitable actions for recovering DIS quality, once it detects a relevant change.

The analysis of DIS modifications arrives to: (i) a classification of the actions according to the kind of modification and to the part of the DIS they affect, and (ii) some guidelines about how each modification of each element of the DIS affects the resulting quality values. In (ii) we present for freshness management, an important concept: the critical path. For each target node in the quality graph there is path that determines the target freshness. This path may change at any time, but target freshness may be improved only by improving this path´s freshness. In general, in (ii), we say for the different possible changes which elements we should modify and which modification we should apply in order to improve DIS quality.

When the Quality Repair module receives an event from the Change Detection module in the QMS, it knows that DIS quality is not satisfying user quality requirements and it must be repaired. The proposed automatic mechanism for determining the repairing actions consists of rules for obtaining an interpretation of the DIS situation from the received event and statistical meta-information, and other rules for determining a ranked list of repairing actions from the interpretation. The idea is that the QMS provides to the user the ranked list of actions and he decides which action to apply using other tools and information from the QMS for applying it. For example, if he wants to apply an action that substitutes an activity by a more efficient one, he needs to know how much the cost of the activity must be reduced. For this, he can use the meta-information maintained by the QMS and the quality evaluation functionalities.

Interpretations, actions and rules for obtaining them are presented. However, they are presented as an illustration of the possibilities of the mechanism, since the intention is not to give an exhaustive set of interpretations, actions and rules, but to give the framework where they can be defined.

In our mechanism interpretations are derived from statistical information. An important kind of statistical information is how frequently a source changes its quality. We want to remark that this information must be obtained from the history of the *quality models* of the source, and may not be obtained from the history of the *quality values* of the source, in which case the obtained frequency would be irrelevant. This is because quality values at a source may be continuously changing, as in the case of freshness factor, but this does not mean that the source quality has relevant changes. In summary, when we suggest the action of eliminating a source or substituting it by another one, we are taking into account the quality models maintained for it.

The information given to the user (DIS administrator) is, in some cases, rather raw, and the user must process it and also query and/or calculate more information for applying the actions. The QMS provides all the necessary information but it does not give it totally processed.

CHAPTER 7. EXPERIMENTATION

1. Introduction

The main goal of our experimentation is to apply to a study case, the techniques for the construction of source and DIS quality models, as well as the maintenance of their history, such that we show the usefulness of the proposals. The idea is to apply to a simple case the whole cycle that includes the successive measurements of sources data, the construction of the quality models of each source, the construction of the quality models of the DIS taking into account the user quality requirements, and the history analysis. For the experimentation we simulate two data sources that are continuously operating and being updated. We design a DIS that extracts data from them and is queried by the users.

Another goal is to implement a prototype that automates the mentioned process, interacting with the user, who decides when a measurement is executed and when a model is calculated. The techniques for DIS quality evaluation, which are needed for the construction of DIS quality models, are implemented in a previously developed prototype, called DQE [Peralta-06], which is now enriched with the calculation of the accepted configurations, proposed in this work. Our tool for quality measurement and quality models construction is designed to interact with DQE tool.

The study case we use is based on a web data source that is real, but its data is generated for the experimentation because it is not yet operating. We describe it in detail in next section. The other data source we consider is a simple database defined by us. The generation of data is made randomly as well as the inclusion of errors.

In this first phase of the experimentation we work only with accuracy factor.

We have done a previous experimentation with a real case, mainly in the measurement of accuracy factor, in a Data Warehouse of the School of Engineering of our University. This was in the context of a project for quality analysis in multi-source information systems [MSISQuality-07]. The main tasks carried out in the experimentation were, at the sources, the identification of the types of errors to measure, the granularity of measurement, and the measurement implementation [Etcheverry+06]. We also analyzed the evaluation of the DW quality from the quality measured at the sources.

In Section 2 we describe the study case, in Section 3 we present the main functionalities of the tool prototype, in Section 4 we present the experiment execution and results, in Section 5 we present the conclusions we arrive through the experimentation, and finally in Section 6 we present the summary of the chapter.

2. Study Case

A *social network* is a social structure made of nodes (generally individuals or organizations) that are tied by one or more specific types of interdependency. A *social network service* focuses on the building of online social networks for communities of people who share interests and activities. Facebook [Facebook-07] is one of the most widely used in 2007, and in this year it began allowing externally-developed add-on applications. Facebook is undergoing a huge period of growth, with more than 150,000 new users signing up daily.

Our study case is based on an add-on application developed for Facebook. This application provides an environment for people to play games together. The users agree meetings for playing games on the web. After they play they have the possibility of rating the game. Given the enormous quantity of users that participate in Facebook proposals, it is expected that hundreds of users will be using

this application daily. This fact has as consequence the continuous growth and change of the application data. For this reason we find the database of this application, called *Fogojogo*, a very interesting data source for our study.

People who want to choose a game for playing or someone who dedicates to creating games for the internet, would find very useful to have trustable information about users' preferences on the existing internet games. We thought about a system that provides this facility to users, extracting data from different sources.

We propose a very simple DIS in order to focus the experimentation on quality changes management, not complicating it with other management problems. The DIS has two data sources, Source1, called *Fogojogo*, which is the real database described above, and Source2, called *Ratings*, which is a data source that provides qualifications for games given from web users. The DIS mainly allows querying about games ratings, integrating information coming from both sources.

Fogojogo is a real database that will be soon operating, but as it is not yet working we decided to simulate its behavior, generating data for it. *Ratings* is a simple data source totally simulated by us.

2.1 Sources

In the following we describe the schemas of the data sources and the mechanisms employed for their population.

2.1.1 Source1 – Fogojogo

Fogojogo is a database containing 5 tables, which store information about games, users' meetings for playing games and users' opinions about games. We consider this schema, which is a part of the original one, in order to manage a smaller schema, not affecting the experimentation possibilities.

The following are the description of the tables:

- **fj_games (name, creator_uid, rates_quantity, rating, last_update)**
  Table *fj_games* stores data about games. Attribute *id* is a number that identifies the game, *name* is the name of the game, *creator_uid* is an identification of the author of the game, *rates_quantity* is the quantity of users that gave an opinion for the game, *rating* is the sum of all the points given by the users for the game, and *last_update* is the last update of the tuple.
- **fj_meetings (id, game_id)**
  Table *fj_meetings* stores data about the meetings for a game. Attribute *id* is a number that identifies the meeting, and *game_id* is the identifier of the game.
- **fj_group_games (meeting_id, uid)**
  Table *fj_group_games* stores data about the users that participate in the meeting. Attribute *meeting_id* is the meeting identifier, and *uid* is the identifier of user.
- **fj_ratings (meeting_id, uid, rated, date)**
  Table *fj_rating*, given a meeting, indicates for each player of the meeting, if he has given an opinion of the corresponding game. Attribute *meeting_id* is the meeting identifier, *uid* is the user identifier, rated indicates if the user evaluated the game or not, and date is the last update of the tuple.
- **fj_users (uid)**
  Table *fj_users* contains all the user identifiers.

In the real case, information about the users is found in the database of Facebook, and the identifiers used here are references to this database.

Figure 7.1 presents a diagram of the relational schema, where primary keys and relations between tables through foreign keys are shown.

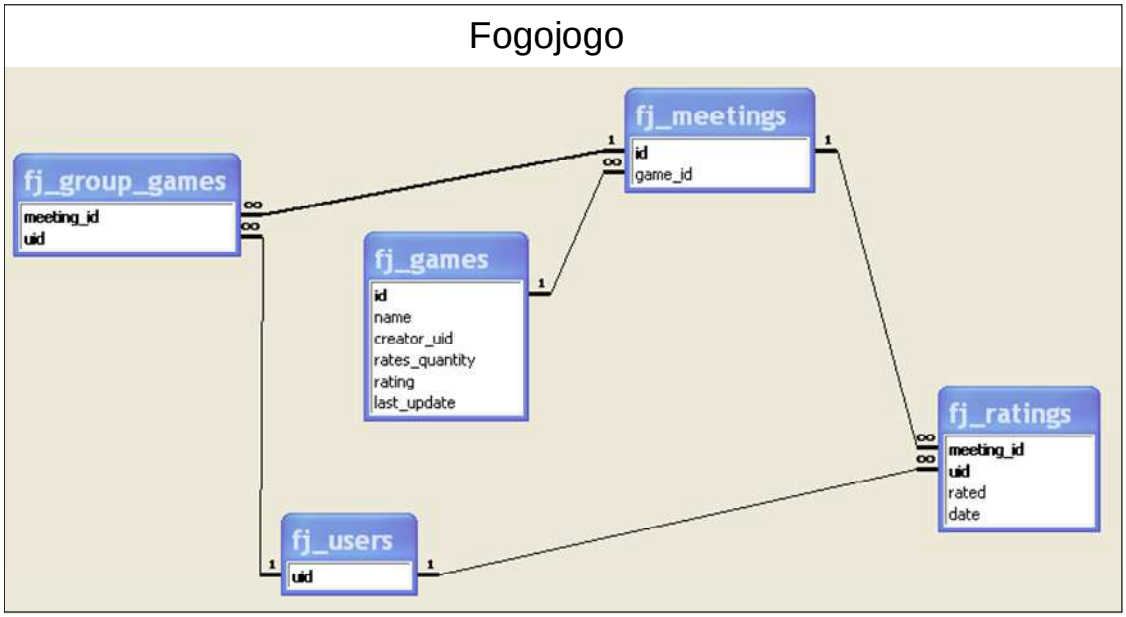


Figure 7.1: Source1 schema

Data generation is done simulating reality, generating randomly the users that agree a meeting, the users that rate a game and the ratings they assign to it. In addition, errors in data are also generated randomly. Data errors are introduced in table *fj_games*, attributes: *name*, *creator_uid*, *rates_quantity* and *rating*.

The following is a high-level pseudo-code of the algorithm for generating Source1 data.

- Generate all games in table *fj_games* with:
  - rate_quantity and rating attributes with value 0
  - initial date for last_update attribute
  - errors: randomly generated in name and creator_uid attributes
- Generate all users in *fj_users*
- Repeat
  - Generate a new meeting in *fj_meetings*
  - For the new meeting generate random quantity of tuples in *fj_group_games*
  - For each new tuple of fj_group_games
    - Generate new tuple in *fj_ratings* where:
      - rated attribute is randomly equal to 0 or 1
    - If rated = 1
      - Update in *fj_games* attributes rates_quantity and rating, corresponding to the game of the current meeting, where:
        - errors are randomly generated
        - if rating has no error, a valid number is randomly generated

Our data generator is capable of generating successively database images. It receives a date and it generates a new image considering the last generated image. This allows us to simulate the data source as a live entity.

2.1.2 Source2 – GamesRatings

Ratings is a database that stores information about users' qualifications of games that are available for playing in internet. We define a very simple version that consists of the following table:

- **ratings (id, uid, game_name, points, date)**
  Table *ratings* stores users' ratings for games. Attribute *id* is a number that identifies the user rating, *uid* is an identification of the user, *game_name* is the name of the rated game, *points* is the rating assigned by the user to the game, and *date* is the date of the user rating.

Data is generated randomly. For each new tuple, attribute *date* has the value of the last date plus a random quantity of minutes. Errors are introduced randomly in data for attributes *game_name* and *points*.

Successive data images are also generated for this data source.

2.2 Data Targets

In *DataTarget1* the DIS provides the average rating of a game and the oldest date that corresponds to this rating. Figure 7.2 shows the data processing graph.

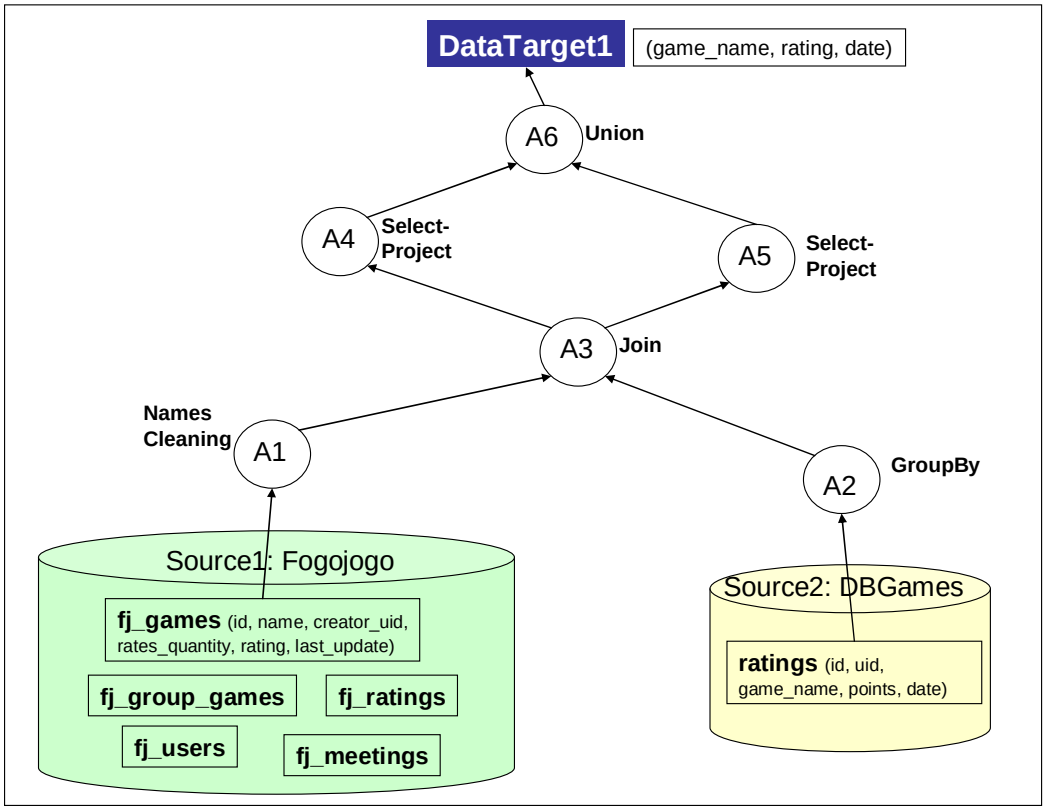


Figure 7.2: Data processing for DataTarget1

The descriptions of the activities of the transformation process are the following:

- **A1: Names Cleaning**
  This activity performs a cleaning on the attribute "name", which corresponds to names of

games. Each value of the attribute is cleaned through a comparison to a referential table of game names. When an invalid value is found, according to a distance metric, the value is substituted by a valid one if there is a certain maximum distance between them, otherwise, it is not modified. In the case the value is modified it is verified if the new value is already in another tuple, and in this case the two tuples are merged.

- **A2: GroupBy**
  This activity performs the following SQL query over Source2.ratings table:

  SELECT game_name, SUM(points) AS points, MAX(date) AS date
  FROM ratings
  GROUP BY game_name

- **A3: Join**
  This activity performs a SQL join between A1 result and A2 result, as follows:

  SELECT *
  FROM A1, A2
  WHERE A1.name = A2.game_name

- **A4: Select-Project**
  This activity performs the following SQL query over A3 result:

  SELECT game_name, ((rating/rates_quantity) + points)/2 AS rating, last_update AS date
  FROM A3
  WHERE last_update < date

  It averages the ratings that come from Source1 and Source2.

  The desired date for each constructed tuple is the smallest one since the freshness of the tuple should be considered depending on the value that was updated before. For selecting the smallest date we use also activity A5. A4 query selects the tuples where attribute *last_update* value (date coming from Source1) is smaller than attribute *date* value (date coming from Source2).

- **A5: Select-Project**
  This activity performs the following SQL query over A3 result:

  SELECT game_name, ((rating/rates_quantity) + points)/2 AS rating, date
  FROM A3
  WHERE date < last_update

  It averages the ratings that come from Source1 and Source2.

  It selects the tuples where attribute *date* value (date coming from Source2) is smaller than attribute *last_update* value (date coming from Source1).

- **A6: Union**
  This activity performs a union between A4 and A5 results.

2.3 Quality Management

The quality factor managed in the experimentation is *accuracy*.

In the following we describe how the factor is measured at the sources, how sources quality models are constructed, how quality is evaluated at the DIS and how DIS quality models are constructed.

Source Accuracy Measurement

Accuracy is measured at cell level (for each value of each tuple) and then it is aggregated to tuple level and table level.

In Source1, we measure accuracy in certain attributes of table *fj_games*. Then, after aggregating the values we obtain only one value of accuracy for Source1. For measuring accuracy of games names, we use a referential table that contains all the valid names. We define a distance (quantity of different characters between two names) for deciding if a name has a mistyping error or it does not belong to the referential table. In Table 7.1 we show the measured attributes and the way they are measured.

Attribute	Accuracy measurement
name	If it belongs to the GamesReferentialTable name_accuracy = 1 Elsif it has mistyping error name_accuracy = 0,5 Else name_accuracy = 0
creator_uid	If it belongs to fj_users table creator_uid_accuracy = 1 Else creator_uid_accuracy = 0
rates_quantity	If it is equal to the quantity of users that rated the game rates_quantity_accuracy = 1 Else rates_quantity_accuracy = 0
rating	If it is greater than 10 * quantity of users that rated the game rating_accuracy = 0 Else rating_accuracy = 1

Table 7.1: Accuracy measurement for Source1

The value of accuracy at tuple level is obtained, for each tuple, in the following manner. If the name of the game does not exist (in the referential) the tuple accuracy is set to 0, regardless of the rest of the attributes' values. If the game exists, a weight is assigned to each measured attribute, such that the most important accuracy value is the one of the rating. The aggregation calculation is the following:

```
If name_accuracy = 0
    tuple_accuracy = 0
Else
    tuple_accuracy = name_accuracy * 0.2 + creator_uid_accuracy * 0.1 +
                     rates_quantity_accuracy * 0.2 + rating_accuracy * 0.5
```

At table level we calculate accuracy as the average of the accuracy values of the tuples.

In Source2 we measure accuracy analogously to Source1. In Table 7.2 we show these measurements.

The value of accuracy at tuple level is obtained analogously to the case of Source1. Here, the attribute with highest weight is the one containing the points assigned by each user. The aggregation calculation is the following:

```
If game_name_accuracy = 0 or points_accuracy = 0
    tuple_accuracy = 0
Else
    tuple_accuracy = game_name_accuracy * 0.2 + points_accuracy * 0.8
```

At table level we calculate accuracy as the average of the accuracy values of the tuples.

Attribute	Accuracy measurement
game_name	If it belongs to the GamesReferentialTable game_name_accuracy = 1 Elsif it has mistyping error game_name_accuracy = 0,5 Else game_name_accuracy = 0
points	If it is between 1 and 10 points_accuracy = 1 Else points_accuracy = 0

Table 7.2: Accuracy measurement for Source2

Source Quality Models for Accuracy

When we have many accuracy measurements of a source at different points through time, we calculate the accuracy model for the source, which is the probability distribution of the possible accuracy values. A precision is selected for accuracy values.

We build these models and we maintain the history of these models, as it is proposed in Chapter 4 of this thesis.

DIS Quality Evaluation for Accuracy

For evaluating accuracy in *DataTarget1* we apply the following propagation functions for each activity of the transformation graph:

Consider input_data as the relation that enters the activity and output_data as the relation that results from the activity, and acc(x) as the accuracy of relation x.

- Names Cleaning:
  acc(output_data) = acc(input_data) + 0.1
  The cleaning process is estimated to improve the accuracy of the input relation in a 10%
- Group By:
  acc(output_data) = acc(input_data)
- Join:
  acc(output_data) = acc(input_data_1) * acc(input_data_2)
- Select-Project:
  acc(output_data) = acc(input_data)

- Union:
  acc(output_data) = (acc(input_data_1) * |input_data_1| + acc(input_data_2) * |input_data_2|) / |output_data|

DIS Quality Models for Accuracy

Using the requirements of type *maximum*, *minimum*, *average* or *most frequent*, DIS Quality is evaluated and compared to the requirements. The history of these evaluations is maintained.

Using the requirement of type *probability* given by the user for *DataTarget1*, **DIS Quality Certainty**, which is the probability that our system satisfies the required value, is calculated as described in Chapter 4.

DIS Quality Distribution is calculated for each possible accuracy value of *DataTarget1* (not taking into account the requirements), as described in Chapter 4.

3. Quality Models Manager – The Prototype

The *Quality Models Manager* prototype has three main functionalities: (1) calculating source quality models, (2) evaluating DIS quality, comparing to quality requirements, and (3) calculating DIS quality models.

The tool is able to measure accuracy of data sources according to the criteria explained in previous section, to calculate the probability distribution of accuracy values from a set of successive measurements, and to maintain and show the histories of both. It executes the evaluation of the accuracy of the selected data target, according to the criteria presented in previous section. It receives user quality requirements and it shows the satisfaction or dissatisfaction of them. In addition, it is able to calculate DIS Quality Certainty (probability of satisfying a user quality requirement), and also to calculate the probability distribution of DIS quality values. Finally, it manages the history of the DIS models.

3.1 Prototype Architecture and Implementation

The prototype's general architecture is designed according to the Model-View-Controller pattern, which allows separating data (model) and user interface (view), through a third component (controller). The *model* component encapsulates the access and management of data, the *view* component manages the user interface, and the *controller* component processes and responds to user actions and generates changes to the model.

We developed the tool as a web application for practical reasons. Therefore our view is the HTML page, and the cotroller receives user actions and acts on the model, which manages a database that stores the system metadata and the sources databases. Finally the model prepares information to be shown by the view. Figure 7.3 shows the general architecture of the tool.

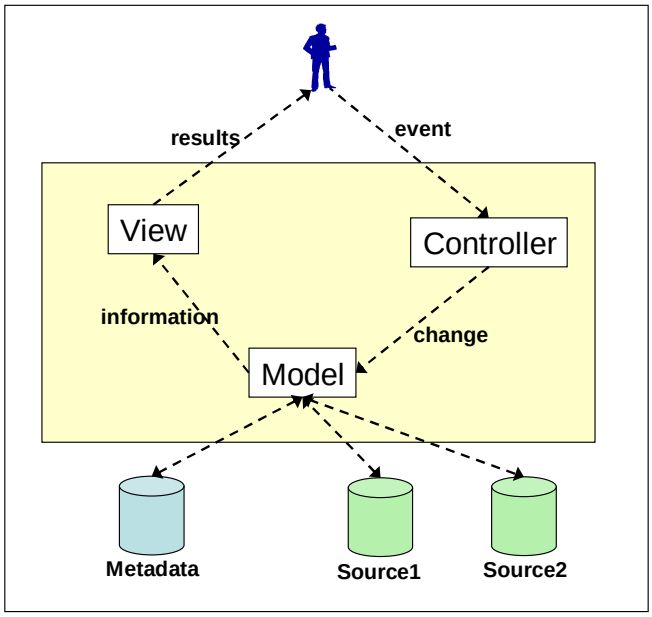


Figure 7.3: Model-View-Controller Architecture

The *Controller* module implements the different tool functionalities, through five modules: *Measurement*, which measures accuracy of sources data, *Distribution Calculation*, which calculates the quality models of sources data, *DIS Certainty Calculation* and *DIS Quality Distribution*, which calculates DIS Certainty and DIS quality distribution, interacting with *Evaluation* module. *Evaluation* module implements the propagation of DIS quality values from the sources to the data targets and also the propagation of quality requirements from the data targets to the sources. This module is totally implemented in this prototype but in later versions it will connect to *DQE* prototype (mentioned before), using its functionalities for quality evaluation. See Figure 7.4 for *Controller* architecture.

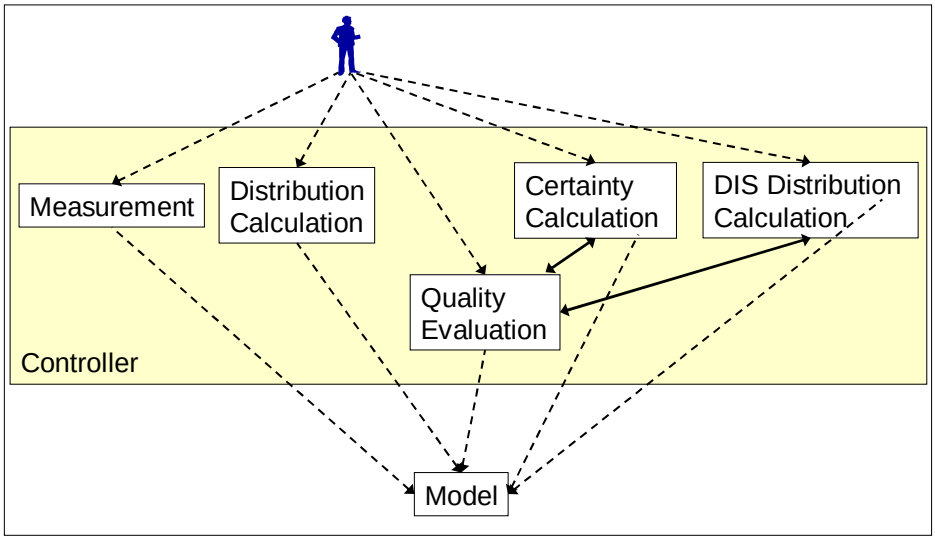


Figure 7.4: Controller Architecture

The prototype is implemented in PHP language and uses MySQL as database management system for the metadata.

3.2 Tool Functionalities

In this section we show the functionalities of the tool and the way it interacts with the user.

3.2.1 Source Models Calculation

After selecting a source and a quality factor, the user must choose among the following actions: *Execute Measurement*, *History of Measurements*, *Distribution Calculation*, *History of Distributions* (see this in Figure 7.5-(a))

Execute Measurement functionality asks the user for the current date and then executes the measurement of the quality factor on the entire source. It then shows the obtained quality values for each cell, for each tuple, and the calculated quality value for the whole source (Figure 7.5-(b)).

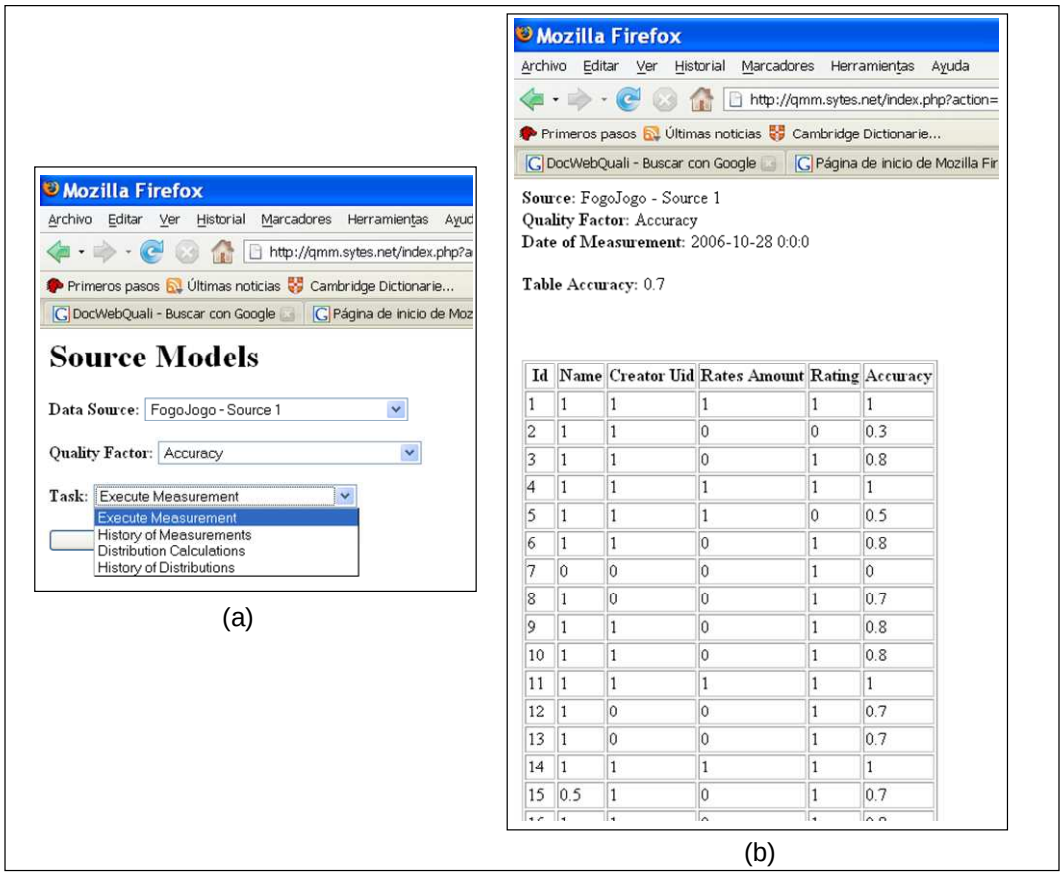


Figure 7.5: (a) Options for source models management, (b) Result of a source measurement

History of Measurements allows selecting from a list containing all the measurement dates, a starting date and an ending date, and then it shows the list of obtained quality values and optionally it shows them graphically. See an example in Figure 7.6.

Distribution Calculation obtains the histogram and the probability distribution and indicators, from the measurements that were executed during a period selected by the user. The user can re-calculate the last calculated distribution, adding new measurements, or he can calculate a new distribution with the new measurements.

History of Distributions functionality gives the history of all the distributions calculated. The evolution of the distribution indicators can be observed graphically. See an example in Figure 7.7.

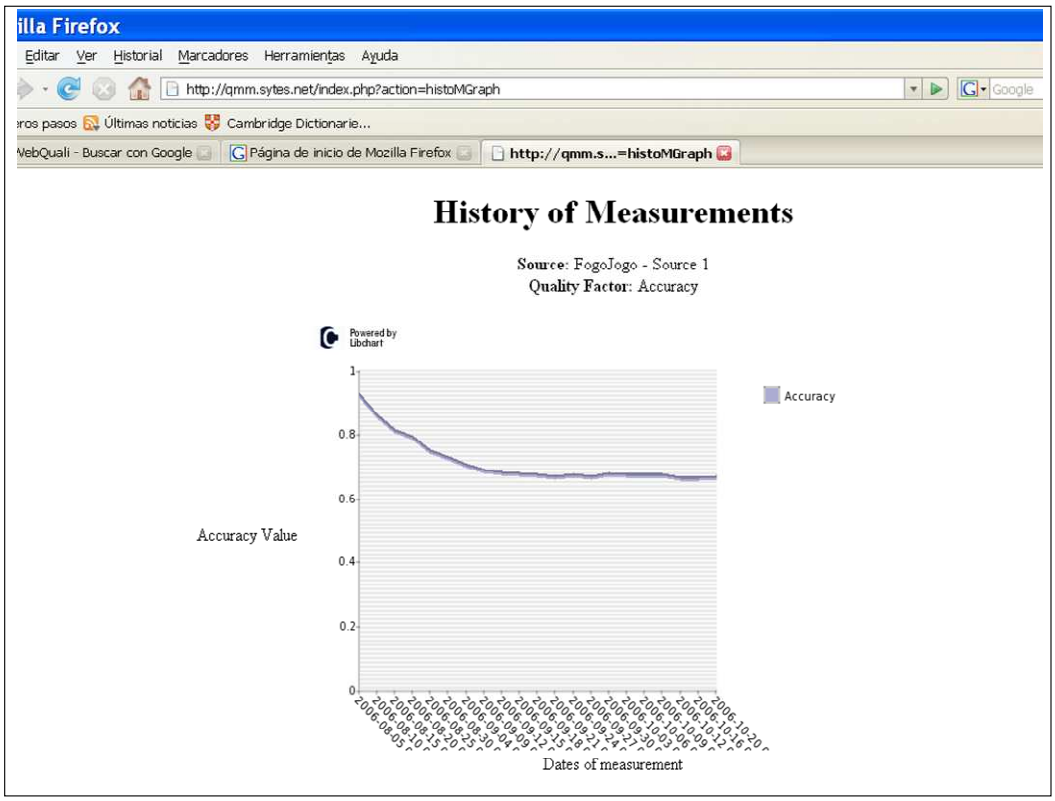


Figure 7.6: History of Measurements

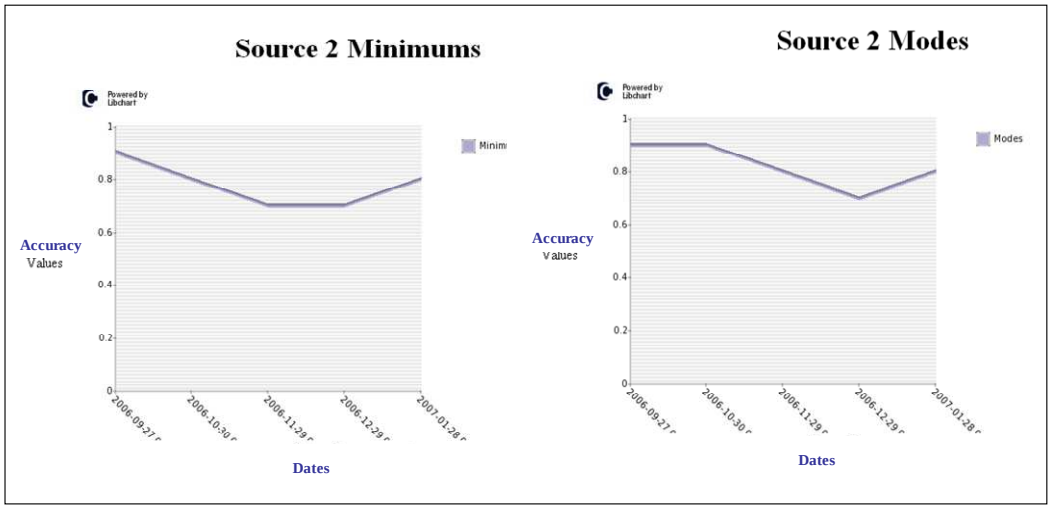


Figure 7.7: History of Distributions – Graphs of minimums and modes

3.2.2 DIS Quality Evaluation

The system receives the user quality requirements that may be of type *maximum*, *minimum*, *average* or *most frequent*, and then it evaluates the corresponding quality values given by the DIS. For example if the user enters two requirements, one for the minimum and another for the average, then the minimum and average values given by the DIS are calculated.

This evaluation is made for certain target; in our case for *DataTarget1*.

3.2.3 DIS Models Calculation

This functionality has three options: *DIS Quality Certainty*, *DIS Quality Distribution*, and *History of DIS Quality*.

DIS Quality Certainty calculates the probability of satisfying the current user quality requirements of type *probability*. For the calculation it uses one of the previously calculated distributions of *Source1* and one of the previously calculated distributions of *Source2*, which are chosen by the user. For obtaining the current DIS Certainty, one should choose the last distributions of the sources.

DIS Quality Distribution calculates the probability distributions of the possible accuracy values of the data target. Analogously to DIS Certainty functionality, the user chooses the sources distributions that are considered.

History of DIS Quality shows all the previously calculated DIS Quality Certainty or DIS Quality Distributions. The evolution of these values is also shown graphically. See an example in Figure 7.8.

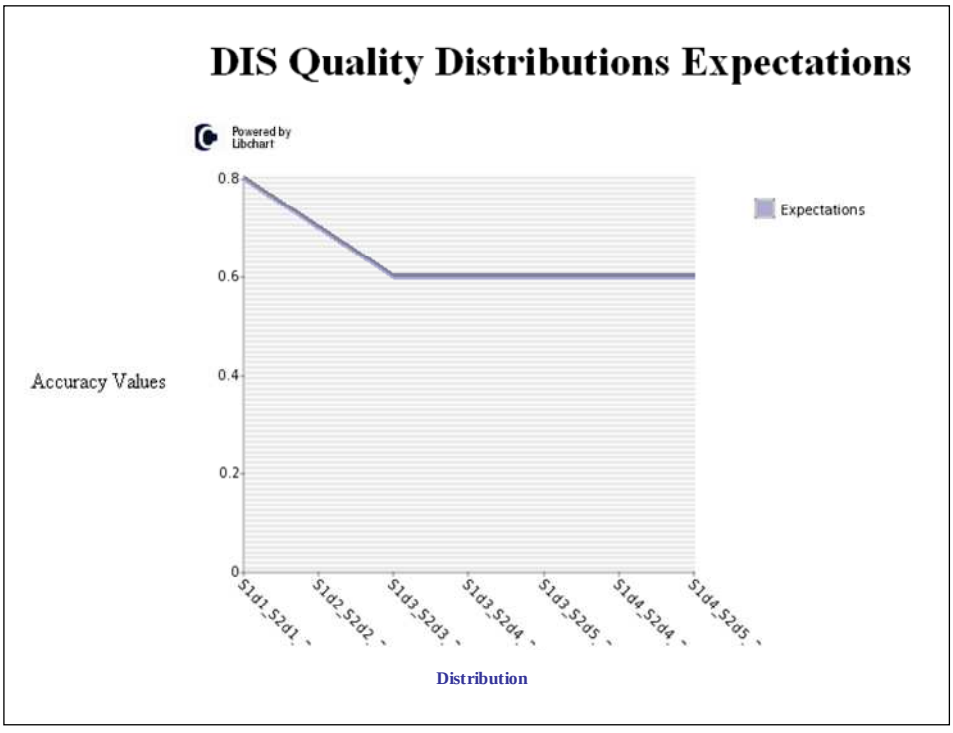


Figure 7.8: History of DIS Quality Distributions – Graph of expectations evolution

4. Experiment Execution and Results

In our experiment we built several quality models of the sources and the DIS. We simulated sources updates during some months (between 4 and 6), and for each source we measured accuracy periodically (approximately each 3 days) and we constructed a quality model each month.

In the simulation we chose between two kinds of data generation: one that generated a low percentage of errors and another that generated a high percentage of errors. The idea was to simulate two kinds of users, an expert one that makes few errors when entering data, and an inexpert one that very frequently makes errors.

We modeled accuracy of *Source1* during approximately 4 months. In the simulation data was entered by an expert user, except in the last month, where during 9 days, an inexpert user was entering data.

We show in Figure 7.9 the graph of the accuracy values that were measured each few days. Then observe, in Figure 7.10 how the models indicators, *minimum* and *mode* evolved during the same period. Both were stable in the first models and in the last one they decreased.

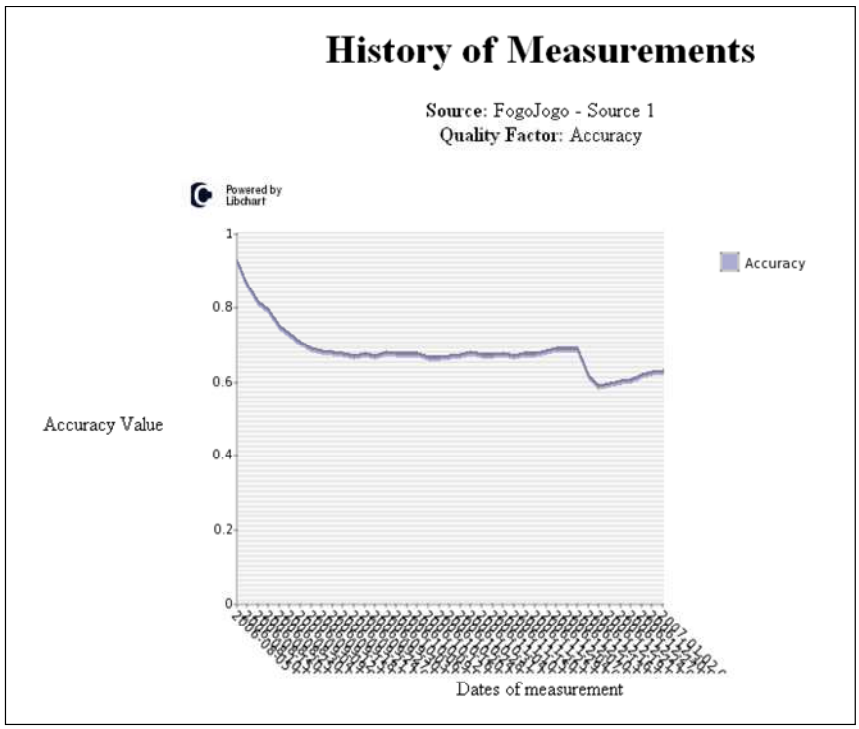


Figure 7.9: History of measurements of *Source1* accuracy

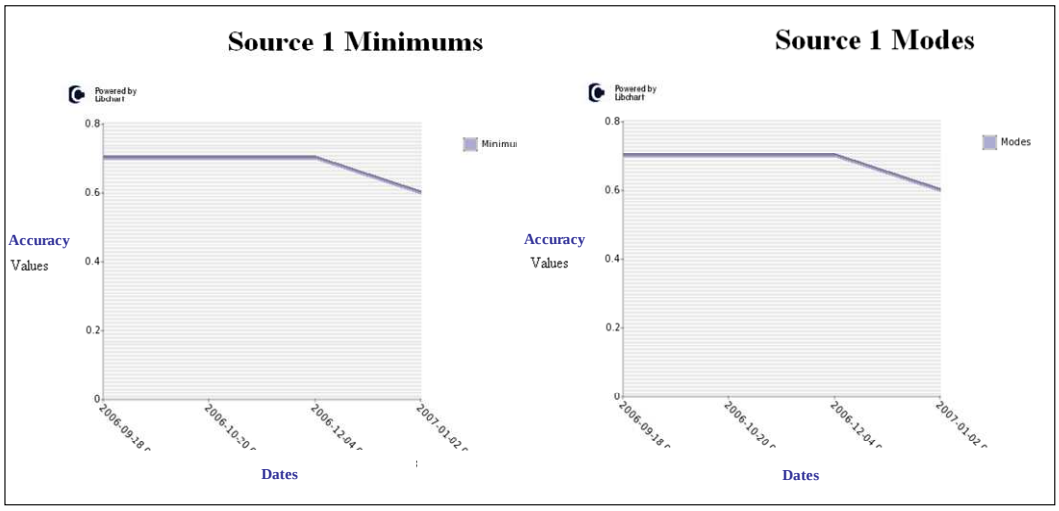


Figure 7.10: History of Distributions of *Source1* – Graphs of minimums and modes

In Figure 7.11 we can observe the evolution of the indicators *maximum* and *expectation*.

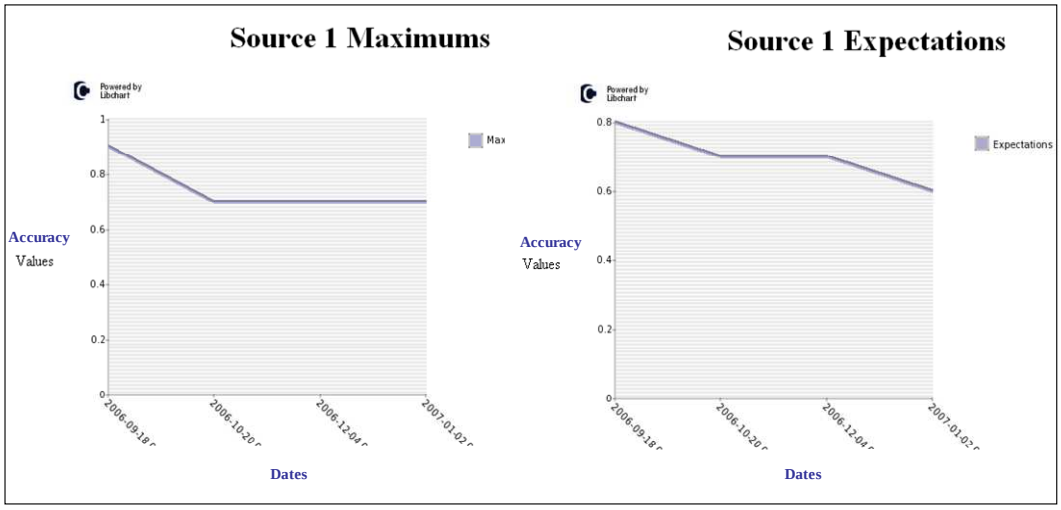


Figure 7.11: History of Distributions of *Source1* – Graphs of maximums and expectations

The evolution of the distribution indicators shows that during the first month the maximum and expectation values decrease, while minimum and mode maintain the same values.

As can be observed, as a consequence of 9 days with the inexpert user, the minimums, expectations and modes decrease drastically.

We modeled accuracy of *Source2* during approximately 6 months. During the middle two months the inexpert user entered the data. In Figures 7.12 and 7.13 we show the obtained distributions graphically.

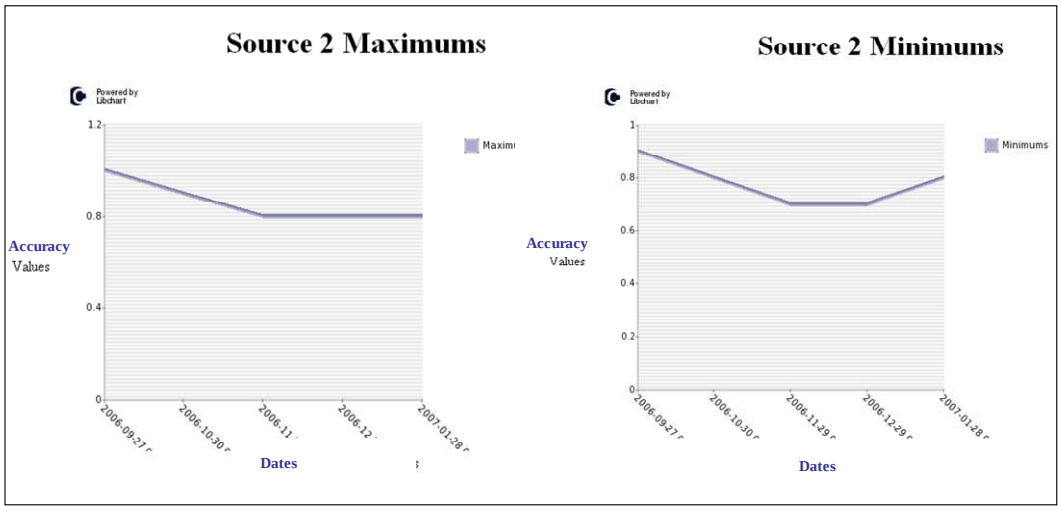


Figure 7.12: History of Distributions of *Source2* – Graphs of maximums and minimums

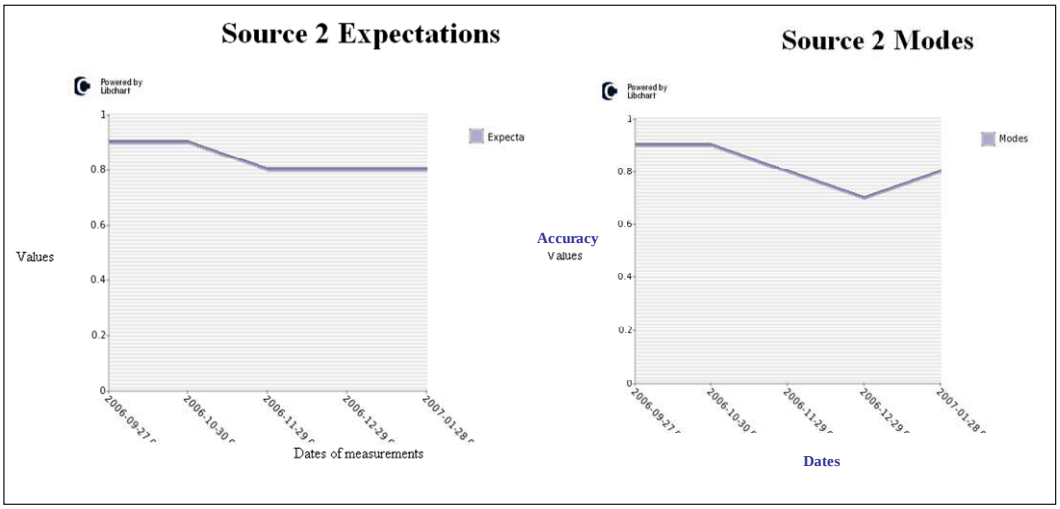


Figure 7.13: History of Distributions of *Source2* – Graphs of expectation and modes

Models of DIS Quality were calculated using the models of sources quality.

DIS Quality Certainty was calculated at different points in time. First, we stated the following requirement:

Minimum Accuracy: 0.8 **Probability: 0.8**

Later, after verifying the low value obtained for Certainty, we stated the following requirement:

Minimum Accuracy: 0.7 **Probability: 0.8**

In each of these calculations the corresponding source quality distributions were used, according to the moment of the calculation. The obtained results are shown in Figure 7.14. In the table given by the tool each row corresponds to a DIS Certainty calculation. Each calculation corresponds to certain distribution of each source, shown in the first two columns, and to certain user quality requirement. The columns "required accuracy" and "with probability" correspond to the user

quality requirement; they give the **minimum** accuracy required and the probability accepted for this minimum. The column Certainty gives the probability that the DIS satisfies the required minimum accuracy. It should be equal or greater than the probability given by the user, for satisfying the user quality requirement.

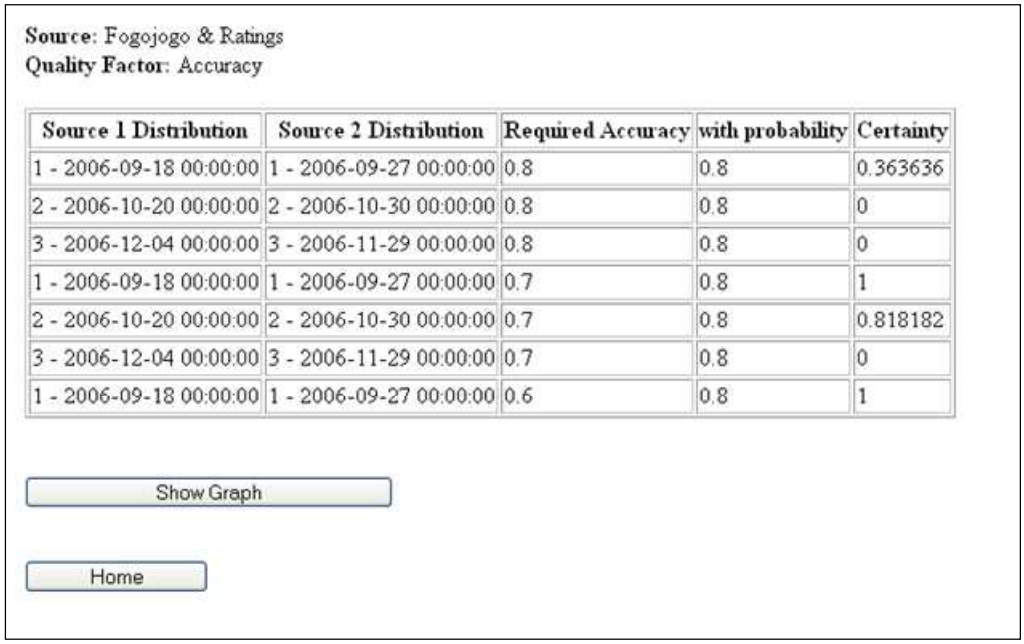
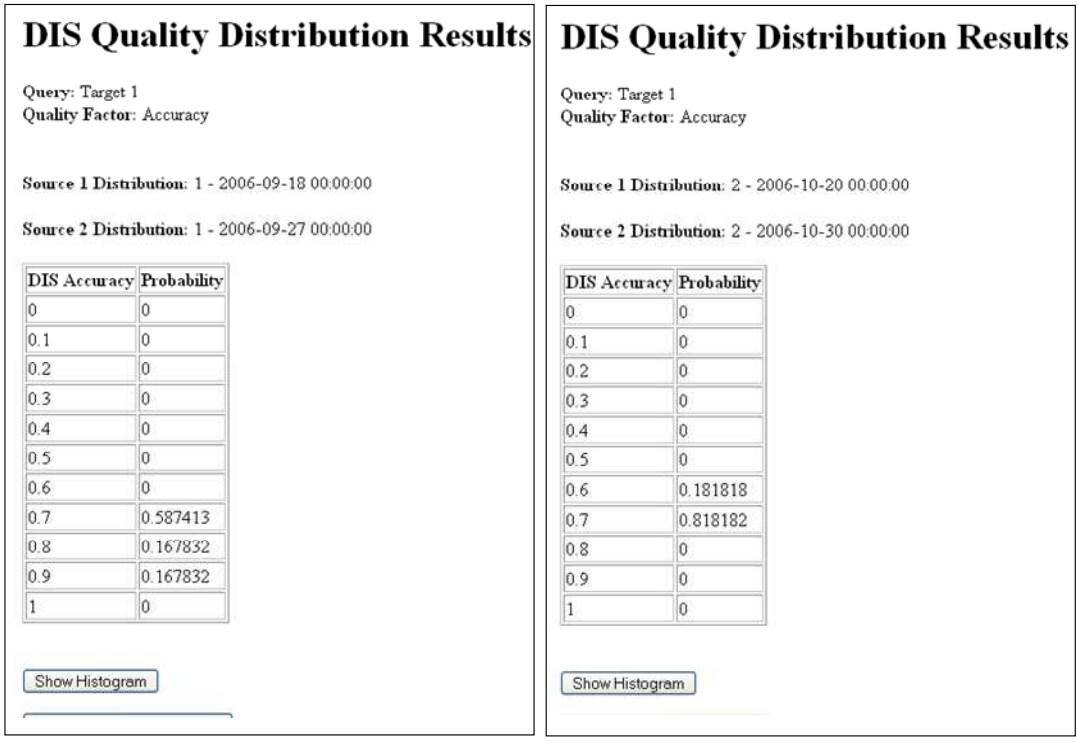


Figure 7.14: DIS Quality Certainty history

As it can be observed, the DIS Certainty has decreased through time. With the first sources' distributions there was a probability of 0.4 for accuracy ≥ 0.8, and 1 for accuracy ≥ 0.7. This happens despite *Source1* minimum was around 0.7, because there is a cleaning task in the process that improves significantly the source's data accuracy. With the following sources' distributions the probability for accuracy ≥ 0.8 and also for accuracy ≥ 0.7 decreases, because the minimum of *Source2* distribution decreases successively.

The probability distribution of DIS quality was also calculated at different points in time. In Figure 7.15 we show two examples: the distribution corresponding to the first sources distributions and the distribution corresponding to the second sources distributions. As can be seen, these results are consistent with the obtained in the Certainty calculation.

In Figure 7.8, previously shown, the history of the indicator *expectation* of DIS distributions is presented. Here, we can observe that at first expectation decreases and then it maintains stable.



DIS Quality Distribution Results

Query: Target 1
Quality Factor: Accuracy

Source 1 Distribution: 1 - 2006-09-18 00:00:00

Source 2 Distribution: 1 - 2006-09-27 00:00:00

DIS Accuracy	Probability
0	0
0.1	0
0.2	0
0.3	0
0.4	0
0.5	0
0.6	0
0.7	0.587413
0.8	0.167832
0.9	0.167832
1	0

Show Histogram

DIS Quality Distribution Results

Query: Target 1
Quality Factor: Accuracy

Source 1 Distribution: 2 - 2006-10-20 00:00:00

Source 2 Distribution: 2 - 2006-10-30 00:00:00

DIS Accuracy	Probability
0	0
0.1	0
0.2	0
0.3	0
0.4	0
0.5	0
0.6	0.181818
0.7	0.818182
0.8	0
0.9	0
1	0

Show Histogram

Figure 7.15: DIS Quality Distributions

5. Experimentation Conclusions

The experimentation allowed us to make some observations and analyze them. In the following we comment this analysis and we present some conclusions we arrived to.

5.1 Results Analysis

With respect to the source quality models, we could observe some characteristics of data sources accuracy behavior. Sources accuracy behaviors are very dependent on the way the source is updated. In the case of *Source1*, where we consider quality of the table *fj_games*, the source updates result in an update on the attribute *rating* of *fj_games* (for the corresponding game). Therefore, when new data has lower accuracy than previous one, it immediately impacts the source accuracy. In contrast, in the case of *Source2*, where updates consist of insertions of new tuples, the decrement of the accuracy in new data does not immediately impact on the source accuracy. A great percentage of new tuples with lower accuracy must be inserted for impacting the accuracy of the whole source. This happens because the accuracy of a table is calculated as the average of the accuracies of its tuples. This characteristic of the behavior is shown in the experiments, since in *Source1* the quality of new data is decreased for nine days and it impacts the quality models, while in *Source2* the quality is decreased for two months and then it impacts the quality models. On the other hand, we observe that in *Source1*, at the beginning, accuracy decreases. This is because at the beginning the games were not rated by anybody and then they start being rated, so mistakes start being introduced. Then they maintain stable if we do not change the kind of data generation.

The calculated DIS quality models show that the probabilities for DIS accuracy values are quite lower than the probabilities at the sources. This is completely reasonable, since for verifying a quality condition at DIS level the corresponding conditions at sources level must be verified simultaneously. If there were more sources in the system, the probabilities of the DIS would be even lower. This fact leads us to think that a DIS with many sources needs very high probabilities of sources verifying the conditions, for having an acceptable level of requirement satisfaction.

Observing the calculated DIS Certainty and DIS probability distributions, we note that *Source2* has more incidence than *Source1* in the quality given by the DIS. We think this is because *Source2* continuously increases its quantity of rows, while *Source1* maintains stable in its quantity of rows.

Due to the considered reality and to the way data was generated, the accuracy values presented by the sources through time did not vary very much (except when we explicitly changed the level of errors generated). Therefore, the calculated distributions had only two or three possible accuracy values.

5.2 Conclusions

This experimentation was a first attempt to applying the proposed techniques. We believe there is much more to do with respect to experimentation. For time limitations reasons, we designed a very limited experiment, where we have only two data sources, only one data target and only one quality factor. We should extend these aspects and also increment the number of executions of the prototype, in order to arrive to more significant results. On the other hand, although the generation of data was done very carefully, generating data and errors randomly, we should do the experimentation with real data.

DIS Quality Certainty calculation turned into a very heavy algorithm; the quantity of operations it implies quickly explodes. When executing this functionality in the prototype, sometimes some minutes are required and in some occasions it cannot arrive to an end. This would get worse if we had, for example, more data sources. This is an important limitation that must be taken into account for future improvements or changes on the proposal. Nevertheless DIS Quality Certainty may be calculated from the calculation of DIS distribution.

DIS Quality Certainty calculation has not the previously commented problem if we work with freshness quality factor. In this case, as it is explained in Chapter 4 of this thesis, the calculation is much simpler. For time limitations reasons we did not run the experimentation with freshness factor.

In spite of being a very primary experimentation, we believe that it was interesting and useful. It materialized the proposals about the quality models, showing their feasibility and also their usefulness. The latter was verified mainly when we discovered characteristics of quality behavior that we had not perceived before the experimentation.

This experimentation did not include all the proposals of the present thesis. Quality change detection and quality repair were not addressed. We plan to extend it including these techniques.

6. Summary

In this chapter we presented the experimentation we have done in order to apply the techniques proposed in the present thesis.

We selected a study case, which was a DIS containing two data sources. Data was successively generated for both sources so that their states through time were simulated. The generation was made with random data and random errors in it. However, the error margins were managed so that two kinds of users entering data were simulated; an expert user and an inexpert one. A data target was defined and also the transformation applied to source data for obtaining it.

Our tool prototype allowed us to successively measure accuracy at source data and to construct models of accuracy behavior for each source. It also allowed calculating DIS accuracy models: DIS Quality Certainty, given certain requirement, and DIS Quality Distribution.

We measured sources accuracy during some months, each few days, and we calculated their models each ten or more measurements. We calculated DIS models considering different pairs of sources distributions.

We were able to make some observations and analysis that were presented in the last sections of this chapter. In previous section we presented some conclusions that we could extract from the experimentation.

CHAPTER 8. CONCLUSIONS

1. Summary

This thesis presents a proposal for maintaining quality in a DIS. The proposal is based on the construction and maintenance of quality behavior models, quality change detection and determination of actions for repairing the quality of the DIS. The most specific parts of the solutions are given for two quality factors: freshness and accuracy.

The most important support of the proposal are the quality behavior models. Thanks to them, the quality changes detection mechanism is able to filter punctual changes that are not important for DIS quality, and in consequence, repair to the DIS is only applied when changes on quality *behavior* have occurred, avoiding unnecessary impacts on the DIS (which later would probably be reverted).

Quality behavior models are probabilistic models of the quality values presented by the data. We propose models for quality of source data and models for quality of data provided by the DIS.

A source quality model is a probability distribution of the quality values of the source. This distribution provides useful indicators of source quality, such as expectation and mode. In the case of freshness factor, the distribution is deduced from information about source data updates. According to the available information, different techniques may be applied for constructing the model. In the case of accuracy factor, the distribution is calculated from a set of measurements of accuracy over the source data. The idea is that after an appropriate number of measurements that have been done on source data over time, the distribution is calculated. For any quality factor, each time a new distribution is calculated, the previous one (its indicators) is stored with the history of the source models.

DIS quality models show the behavior of the targets-data quality given by the DIS. We characterize DIS quality, through two different approaches, as: the quality values that the DIS can take, and the quality considering the user quality requirements, i.e. the satisfaction or not of the quality requirements. The DIS model may be given through three different aspects: (1) The *DIS Quality Certainty*, which is the probability that the DIS satisfies the quality requirements that have a probability associated, (2) the information about the satisfaction of the requirements of maximum, minimum, average and most frequent value, and (3) the probability distribution of the possible quality values satisfied by the DIS. (1) and (2) are calculated through the application of probabilistic techniques, while (2) consists on the evaluation of DIS quality from the sources models' indicators and their comparison to the quality requirements.

The management of probabilities for the quality values in the system allowed us to propose a variety of types of user quality requirements, which gives expressiveness to the user and allows him to pose more flexible requirements. For example, the user may state as quality requirement the average quality value he wants to obtain in certain query.

The mechanism proposed for quality changes detection is based on events that are managed through rules. There are events that come from the sources to the DIS and events that are generated at the system. The latter are generated as a consequence of a change on a source quality model, a change in the transformation graph or a change in the user quality requirements. Rules process these events and eventually generate new ones. With the successively application of rules the selection of the real relevant changes is achieved. When a relevant change is detected an event is generated to be captured by the quality repairing module. The defined events and rules are based on a classification of changes according to how they affect the DIS.

For DIS quality repairing we propose to analyze the situation of the DIS beyond the occurred change and the current conditions of the system. We are able to achieve this because we take advantage of the quality models, their history, and information about other properties of the system that affect the quality factor. For example, if we are analyzing a problem with freshness of data given by the DIS, we may consider the cause of the freshness behavior change, the history of freshness behavior in this data, and the history and current values of the activities' costs of the transformation graph. The analysis of DIS situation has as result an *interpretation*, and then the possible suitable *repairing actions* are deduced from it. Rules are proposed for the deduction of interpretations and actions. A ranked list of possible repairing actions is given as final result of this process.

We did an experimentation with the purpose of applying some of the proposed techniques to a study case the most close to reality as possible. We chose as application domain the playing and evaluation of games in internet by a user community. We based our case on a real data source, but we generated data for both considered data sources. Data sources were simulated through time. A tool prototype was implemented, which allowed us to apply the proposed techniques for quality models construction and maintenance. Some interesting observations about sources quality behavior and DIS quality behavior could be made.

2. Contributions

The main contributions of this thesis are the following:

- Techniques for modeling quality behavior in a DIS
  We propose to build and maintain probabilistic models of the quality factors at the sources and DIS. We provide techniques for modeling quality of the sources for freshness and accuracy, which apply to different scenarios. We also provide techniques for modeling quality of the DIS for freshness and accuracy factors.
- A mechanism for detecting relevant quality changes in a DIS
  We provide a mechanism for detecting changes of DIS quality from events that notify certain changes in different DIS elements. The main advantage of the mechanism is that it filters a lot of changes that are not relevant and selects only the changes that deserve a treatment, which are the ones that generate the dissatisfaction of user quality requirements. After processing the events and evaluating the effects of the changes it notifies only the relevant changes.
- A mechanism for analyzing DIS situation and finding the most suitable actions for recovering quality
  We provide a mechanism that from the occurred relevant changes analyzes the situation of the DIS, basing on statistical information maintained in the management system, deduce an interpretation of the situation and then determines a ranked list of actions for recovering DIS quality. The mechanism basically consists of sets of rules, which can be extended adding new rules, new interpretations and new actions. The provided interpretations, actions and rules show the usefulness of the mechanism.

3. Concluding Remarks

Our proposal can be seen at two different levels of abstraction. The first level has the advantage that its generality allows it to be applied to any quality factor. The second level consists of solutions that are specific for freshness and accuracy quality factors. With respect to *quality models*, in the case of

sources' ones, the proposal of constructing and maintaining the models is general, may be applied to any quality factor, while the proposed mechanisms for constructing the models is specifically for freshness and accuracy. In the case of DIS quality models, we believe the whole proposal may be applied to any factor, taking into account that there is a differentiation between two kinds of factors, which are treated differently. With respect to *quality change detection*, the proposed mechanism as well as the defined events and rules are general, they can be applied to any quality factor. Perhaps some modification may be done to events' attributes, adding information that is particular to a quality factor, for passing it to the quality repairing module. Finally, with respect to *DIS quality repair*, we consider that the proposed mechanism and the structure of the rules are general, while the interpretations, statistics, actions and the particular rules presented are specific to freshness and accuracy factors. Table 8.1 summarizes this analysis of the proposal.

We believe that the main advantage of this proposal is that it leads to the application of a **preventive strategy**, at the same time **optimizing the corrective actions** that are applied to the DIS for maintaining quality.

The use of probabilistic models for the quality of the DIS allows acting preventively, since actions are taken if the possibility of not satisfying the required quality increases, for example if DIS Quality Certainty decreases from 0.9 to 0.7. This means that in some cases actions are carried out before quality values obtained by the users effectively get worse, so avoiding quality changes.

		General	**Specific to quality factor**
Source Quality Models	To construct the models	X	
	How to construct the models		X
DIS Quality Models	To construct the models	X	
	How to construct the models	X	
Quality Changes Detection	The mechanism	X	
	Events and rules	X	
DIS Quality Repair	The mechanism and rules structure	X	
	Interpretations, statistics, actions, particular rules		X

Table 8.1: Applicability of the proposal of this work

Corrective actions are optimized, since on one hand, they are applied only when DIS quality has demonstrated that it really needs to be improved (because it had a change in its behavior), and on the other hand, they are applied taking into account an analysis of the DIS situation, which is based on a big amount of information and statistics about the DIS.

In this work we propose to take actions when DIS quality behavior does not satisfy user quality requirements; concretely, when DIS Quality Certainty does not reach the probability asked by the user, and when quality indicators: minimum, maximum, average or most probable value, do not reach the ones asked by the user. However, we also propose to maintain the probability distribution of DIS quality values, which is independent of user quality requirements. This information is very useful for doing **tendencies** studies. Tendencies could be detected in the histories of quality models,

and, defining appropriate criteria, we could act for avoiding harmful changes on the DIS. The complexity of this problem resides in the determination of valid criteria for deciding when something is a clear tendency such that it can be assumed that the analyzed fact will continue evolving in the same way.

4. Limitations

Up to now we have detected two limitations in our proposal. The first one refers to the kind of application domains where it can be applied, and the second one refers to the implementation of one of the proposed algorithms.

There may be some application domains that cannot admit even one error, or cannot admit that a quality value passes certain threshold even one time. For example, in the management of data obtained from medical instruments, which give information for determining an exact medicine dose or treatment for a patient. In our proposal, quality behavior is observed and certain quality behavior is assured, but it is not guaranteed that some punctual dissatisfaction of quality requirements will not occur.

The experimentation, which was our first attempt to apply the proposed techniques, has shown that one of the probability calculations we propose, in some cases, is extremely expensive to implement. This happens when we calculate DIS Quality Certainty for accuracy, which is the probability of satisfying certain accuracy value at the DIS, and there are a big number of combinations of sources accuracy values that satisfy this value. In this case, it must be calculated the union of many probabilistic events, and this calculation may be very heavy. This problem gets worse as we increment the quantity of sources of the DIS.

5. Future Work

With respect to the present work, there are some aspects that could be improved or treated more in depth.

The specification of the framework, algorithms and mechanisms may be improved, completing and unifying the models and specification languages. In particular, statistic data was not clearly specified, and some assumptions were made for its utilization in the solutions.

The repairing actions, together with the possible system analysis and interpretations, may be studied more in depth and many more cases may be deduced and specified. An interesting possibility for the problem of quality repairing would be the proposition of patterns for deducing interpretations and determining repairing actions, and a mechanism for their reusing.

Finally, much more experimentation should be done, testing different quality evolutions, and considering completely real cases. The difficulty in this issue is that the real cases for our experimentation must allow us to monitor the data for a period in time, being not enough to have one particular image of data. In addition, experimentation should be extended to the detection and treatment of changes. We currently have different prototypes, one for the management of quality models, and another for quality evaluation; we should integrate them and add the functionalities for change detection and repairing actions recommendations. These two functionalities would be implemented through a logic language that allows defining rules.

With respect to new works related to this one or as a continuation of it, we see the following two main directions:

- The study of other quality factors, and the application and extension of this proposal to them.
  As previously commented, our proposal has parts that are applicable to other quality factors and other parts that are specifically proposed for accuracy and/or freshness factors. In addition, the thesis of V. Peralta [Peralta-06], whose solutions are very much used in our proposal, also focuses on accuracy and freshness. Therefore, we think that there is much work to do if a new quality factor is selected and quality evaluation and quality maintenance techniques are searched for it. We think that our framework can be reused, and also many parts of these both proposals, but at the same time many new challenges will appear and as a secondary effect the existing solutions would be improved and generalized.
- Addressing the problem of quality maintenance considering the relation that exists between the different factors.
  Up to now we considered each quality factor independently. Quality evaluation and maintenance was solved for one quality factor at a time. However, we have detected that they are not independent. Many quality factors are related in their semantics, which means that they inherently affect each other; the value of one factor always affects the value of the other. For example, *availability* affects *freshness*, *freshness* affects *accuracy*, *and accuracy* affects *usefulness*. In addition, many quality factors are related because the actions that improve one factor directly affect the other one. For example, adding a cleaning task to the transformation process that is applied to data, affects the freshness of the resulting data. We believe that quality maintenance considering many aspects of quality at the same time is an important research challenge.

BIBLIOGRAPHY

[Andritsos+06] P. Andritsos, A. Fuxman, R. J. Miller. *Clean Answers over Dirty Databases: A Probabilistic Approach.* Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA. IEEE 2006.

[Antova+07] L. Antova, C. Koch, D. Olteanu. *MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions.* Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, April 15-20, 2007, Istanbul, Turkey. IEEE 2007.

[Ballou+95] D.P. Ballou, H.L. Pazer. *Designing Information Systems to Optimize the Accuracy-timeliness Tradeoff*. Information Systems Research (6:1), Mar. 1995, pp. 51-72

[Ballou+99] D. P. Ballou, G. K. Tayi. *Enhancing Data Quality in Data Warehouse Environments.* Communications of the ACM 42(1): 73-78 (1999)

[Ballou+03] D.P. Ballou, and Pazer H.L. (2003), *"Modeling Completeness versus Consistency Tradeoffs in Information Decision Systems"*, IEEE Transactions fn Knowledge Management and Data Engineering (15:1), Jan.\Feb. 2003, pp. 240-243

[Ballou+06] D. P. Ballou, I. N. Chengalur-Smith, R. Y. Wang. *Sample-Based Quality Estimation of Query Results in Relational Database Environments.* IEEE Transactions on Knowledge and Data Engineering. Vol 18, No 5, May 2006.

[Benjelloun+06] O. Benjelloun, A. Das Sarma, A. Halevy, J. Widom. *ULDBs: Databases with Uncertainty and Lineage.* VLDB 2006

[Bertolazzi+01] P. Bertolazzi, M. Scannapieco. *Introducing Data Quality in a Cooperative Context.* Proceedings of the 6th International Conference on Information Quality (ICIQ'01), Boston, MA, USA, 2001.

[Boulos+05] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, D. Suciu. *MYSTIQ: A system for finding more answers by using probabilities.* Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005. ACM 2005, ISBN 1-59593-060-4.

[Bouzeghoub+00] M. Bouzeghoub, Z. Kedad. *A Logical Model for Data Warehouse Design and Evolution.* DaWaK'00.

[Bouzeghoub+03] M. Bouzeghoub, B. Farias Lóscio, Z. Kedad, A.C. Salgado. *Managing the Evolution of Mediation Queries*. CoopIS/DOA/ODBASE 2003: 22-37

[Bright+02] L. Bright, L. Raschid. *Using Latency-Recency Profiles for Data Delivery on the Web.* In Proc. of the 28th Int. Conf. on Very Large Databases (VLDB'02), China, 2002.

[Bugajski+05-a] J. Bugajski, R. L. Grossman, E. Sumner, Z. Tang. *An event based framework for improving information quality that integrates baseline models, causal models and formal reference models.* Int. Workshop on Information Quality in Information Systems (IQIS), June 2005, USA.

[Bugajski+05-b] J. Bugajski, R. L. Grossman, E. Sumner, T. Zhang. *A Methodology for Establishing Information Quality Baselines for Complex, Distributed Systems.* Proceedings of the 10th International Conference on Information Quality MIT, Cambridge, Massachusetts, USA. November 4 - 6, 2005 (IQ'2005).

[Bugajski+06] J. Bugajski, R. Grossman, E. Sumner, S. Vejcik. *Monitoring Data Quality for Very High Volume Transaction Systems.* Proceedings of the 2006 International Conference on Information Quality. IQ 2006.

[Caballero+04] I. Caballero, O. Gomez, M. Piattini. *Getting Better Information Quality by Assessing and Improving Information Quality Management.* Proceedings of the 9th International Conference on Information Quality MIT, 2004 (IQ'2004).

[Calero+01] C. Calero, M. Piattini, C. Pascual, M. A. Serrano. *Towards Data Warehouse Quality Metrics.* Proceedings of the 3rd Intl. Workshop on Design and Management of Data Warehouses, DMDW'2001, Interlaken, Switzerland, June 4, 2001.

[Canavos-88] G. Canavos. *Probabilidad y Estadística. Aplicaciones y Métodos*. Mc. Graw Hill. ISBN: 968-451-856-0

[Cappiello+06-a] C. Cappiello, P. Ficiaro, B. Pernici. *HIQM: a Methodology for Information Quality Monitoring, Measurement, and Improvement.* In Proceedings of the International Workshop on Quality of Information Systems, in conjunction with the 25th International conference on Conceptual Modeling (ER 2006).

[Cappiello+06-b] C. Cappiello, B. Pernici. *A Methodology for Information Quality Management in self-healing Web Services.* (Completed academic paper, Politecnico di Milano, Milano, Italy). Proceedings of the 2006 International Conference on Information Quality (ICIQ'06, MIT IQ Conference).

[Cappiello-04] C. Cappiello, C. Francalanci, B. Pernici. *A Rule-Based Methodology to Support Information Quality Assessment and Improvement.* Studies in Communication Sciences, vol. 4, no. 2, December 2004, pp. 137-154.

[Cappiello-05] C. Cappiello. *Data Quality and Multichannel Services.* Phd. Thesis. Politecnico di Milano, Dipartimento di Elettronica e Informazione. May 2005.

[Chawathe-04] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. *The TSIMMIS Project: Integration of Heterogeneous Information Sources.* In Proceedings of IPSJ Conference, pp. 7-18, Tokyo, Japan, October 1994.

[Cheng+05] R. Cheng, S. Singh, S. Prabhakar. *U-DBMS: a database system for managing constantly-evolving data.* Demo. 31st Int. Conference on Very Large Data Bases VLDB '05. Trondheim, Norway, August-September, 2005.

[Cho+03] J. Cho, H. Garcia-Molina. *Estimating Frequency of Change.* ACM Transactions on Internet Technology (TOIT), Volume 3, Issue 3 , Pages: 256 – 290. 2003.

[Choobineh-95] J. Choobineh, A. Kini. *SQLSAM: SQL for Statistical Analysis and Modeling.* Proceedings of the 28th Annual Hawaii International Conference on System Sciences - 1995.

[Chu+05] F. Chu, Y. Wang, D. S. Parker, C. Zaniolo. *Data Cleaning Using Belief Propagation.* Int. Workshop on Information Quality in Information Systems (IQIS), June 2005, USA.

[Curry+07] C. Curry, R. L. Grossman, D. Locke, S. Vejcik, J. Bugajski. *Detecting changes in large data sets of payment card data: a case study.* Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Industrial and government track short papers. California, USA, August 2007.

[DaQuinCIS] *DaQuinCIS.* Joint project among: Università di Roma "La Sapienza", Politecnico di Milano, Università di Milano Bicocca. http://www.dis.uniroma1.it/~dq/index.html (last accessed: 14th Oct. 2007).

[Elmasri+00] R. Elmasri, S. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 2000. ISBN 0-8053-1755-4.

[Etcheverry-07] L. Etcheverry. *Medición de calidad en experimentos de expresión génica con microarrays.* EVaBio'07 I Workshop Español sobre Extracción y Validación de Conocimiento en Bases de Datos Biomédicas, en el marco de CAEPIA 2007, 12/11/2007, Salamanca España.

[Etcheverry+06] L. Etcheverry, S. Tercia, A. Marotta, V. Peralta. *Medición de la Exactitud de Datos Fuente: Un caso de Estudio.* Technical Report, InCo, Universidad de la República, Uruguay, 2006.

[Even+05] A. Even, G. Shankaranarayan. *Value-Driven Data Quality Assessment.* Proceedings of the 2005 International Conference on Information Quality (MIT IQ Conference).

[Ferradina+95] F. Ferradina, T. Meyer, R. Zicari. *Measuring the Performance of Immediate and Deferred Updates in Object Database Systems.* OOPSLA Workshop on Object Database Behaviour, Benchmarks and Performance. Austin, Texas, October 15, 1995.

[Ferradina+96] F. Ferradina, S. Lautemann. *An Integrated Approach to Schema Evolution for Object Databases.* OOIS 1996, London, U.K.

[Galhardas+00] H. Galhardas, D. Florescu, D. Shasha, E. Simon. *AJAX: An Extensible Data Cleaning Tool,* SIGMOD 2000 demo paper (extended version).

[Galhardas+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, C. Saita. *Declarative Data Cleaning: Language, Model, and Algorithms.* Proceedings of 27th International Conference on Very Large Data Bases, VLDB 2001. Roma, Italy.

[Gertsbakh-89] I. B. Gertsbakh. *Statistical Reliability Theory*. Probability: Pure and Applied. A Series of Text Books and Reference Books. MARCEL DEKKER, INC., 1989. ISBN: 0-8247-8019-1.

[Hillier+91] F. Hillier, G. Lieberman. *Introducción a la Investigación de Operaciones.* Mc.Graw-Hill. 1991. ISBN 968-422-993-3.

[IOCourse-07] Lecture notes of course: *Introducción a la Investigación de Operaciones*. 2007. Facultad de Ingeniería, Universidad de la República, Uruguay.

[Jarke+97] M. Jarke, Y. Vassiliou. *Data Warehouse Quality: A Review of the DWQ Project*. Invited Paper, Proc. 2nd Conference on Information Quality. MIT, Cambridge, 1997.

[Jeusfeld+98] M. A. Jeusfeld, C. Quix, M. Jarke. *Design and Analysis of Quality Information for Data Warehouses.* ER 1998: 349-362

[Juran-88] J. M. Juran. *Juran's Quality Control Handbook.* 4 ed, ed. F.M. Gryna. 1988, New York: Mc-Graw-Hill, Inc.

[Karakasidis+05] A. Karakasidis, P. Vassiliadis, E. Pitoura. *ETL Queues for Active Data Warehousing.* Int. Workshop on Information Quality in Information Systems (IQIS), June 2005, USA.

[Koeller+02] A. Koeller, E. A. Rundensteiner. *Incremental Maintenance of Schema-Restructuring Views.* EDBT'02

[Lakshmanan+96] L. V. S. Lakshmanan, F. Sadri, I. N. Subramanian. *SchemaSql – A Language for Interoperability in Relational Multi-database Systems.* VLDB'96

[Lautemann-97] S. Lautemann. *Schema Versions in Object Oriented Database Systems*. In proc. of the 5th. Int'l. Conf. On Database Systems for Advanced Applications (DASFAA), Melbourne, Australia, April 1997.

[Lee+00] M. Lee, T. W. Ling, W. L. Low. *IntelliClean: a knowledge-based intelligent data cleaner*. Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, August 20-23, 2000, Boston, MA, USA. ACM, 2000

[Lee+02] Y. Lee, D. Strong, B. Kahn, R. Wang. *AIMQ: A Methodology for Information Quality Assessment.* Information & Management, December 2002, Volume 40, Issue 2, pp. 133-146.

[Liu+04] Z. Liu, C. Luo, J. Cho, W. W. Chu. *A Probabilistic Approach to Metasearching with Adaptive Probing.* ICDE 2004

[Madnick+01] S. Madnick, R. Wang, F. Dravis, X. Chen. *Improving the Quality of Corporate Household Data: Current Practices and Research Directions.* Proceedings of the Sixth International Conference on Information Quality, November 2001. pp. 92-104.

[Madnick+04] S. Madnick, R. Wang, X. Xian. *The Design and Implementation of a Corporate Householding Knowledge Processor to Improve Data Quality*. Journal of Management Information Systems, Vol. 20, No. 3, Winter 2003-04, pp 41-69.

[Marchetti+03] C. Marchetti, M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni. Data *Quality Notification in Cooperative Information Systems.* International Workshop on Data Quality in Cooperative Information Systems. Italy, 2003.

[Marotta+03] A. Marotta, R. Ruggia. *Quality Management in Multi-Source Information Systems.* II Workshop de Bases de Datos. Jornadas Chilenas de Computación. Chile. Nov. 2003.

[Marotta+05] A. Marotta, R. Ruggia. *Managing Source Quality Changes in Data Integration Systems.* Second International Workshop on Data and Information Quality (DIQ'05) (with CAISE). June, 14th. 2005, Porto, Portugal

[Marotta+06] A. Marotta, F. Piedrabuena, A. Abelló. *Managing Quality Properties in a ROLAP Environment.* In Proceedings of 18th. Conference on Advanced Information Systems Engineering (CAISE'06). 5-9 June, 2006. Luxembourg.

[McBrien+02] P. McBrien, A. Poulovassilis. *Schema Evolution in Heterogeneous Database Architectures, a Schema Transformation Approach*. CAiSE'02: 484-499.

[Mecella+03] M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, C. Batini. *The DaQuinCIS Broker: Querying Data and Their Quality in Cooperative Information Systems.* J. Data Semantics I 2003:208-232

[MSISQuality-07] *Analisis de Factores de Calidad en Sistemas de Información Multi-fuentes.* Project at Universidad de la República, supported by Comisión Sectorial de Investigación Científica. Main researchers: A. Marotta, V. Peralta, R. Ruggia. Finished: March, 2007.

[Moraes+07] M. Moraes Batista, A. C. Salgado. *Information Quality Measurement in Data Integration Schemas.* 5th International Workshop on Quality in Databases at VLDB. September 23, 2007, Vienna, Austria.

[Müller+03] H. Müller, J. C. Freytag. *Problems, Methods, and Challenges in Comprehensive Data Cleansing.* Technical Report, HUB-IB-164, Humboldt University Berlin, Berlin, Germany, 2003.

[Naumann+99] F. Naumann, U. Leser, J. C. Freytag. *Quality-driven Integration of Heterogenous Information Systems.* VLDB 1999: 447-458

[Naumann+00] F. Naumann, C. Rolker. *Assessment Methods for Information Quality Criteria.* 5th. International Conference on Information Quality, 2000. IQ 2000: 148-162.

[Neely-05] M. P. Neely. *The Product Approach to Data Quality and Fitness for Use: A Framework for Analysis.* Proceedings of the 10th International Conference on Information Quality MIT, Cambridge, Massachusetts, USA. November 4 - 6, 2005 (IQ'2005)

[Nguyen+89] G. T. Nguyen, D. Rieu. *Schema Evolution in Object-Oriented Database Systems.* Data & Knowledge Engineering (DKE) , Volume 4, 1989.

[Nica+98] A. Nica, A. J. Lee, E. A. Rundensteiner. *The CVS Algorithm for view synchronization in evolvable large-scale information systems.* EDBT'98.

[Nica+99] A. Nica, E. A. Rundensteiner. *View Maintenance after View Synchronization.* IDEAS'99

[Oliveira+04] P. Oliveira, F. Rodrigues, P. Henriques. *Limpeza de Dados - Uma Visão Geral*. In Proc. of Data Gadgets Workshop, Malaga, Spain, 2004.

[Oliveira+05] P. Oliveira, F. Rodrigues, P. Henriques. *A Formal Definition of Data Quality Problems.* Proceedings of the 10th International Conference on Information Quality MIT, Cambridge, Massachusetts, USA. November 4 - 6, 2005 (IQ'2005)

[Papastefanatos+07] G. Papastefanatos, P. Vassiliadis, A. Simitsis, Y. Vassiliou. *What-If Analysis for Data Warehouse Evolution.* Data Warehousing and Knowledge Discovery, 9th International Conference, DaWaK 2007, Regensburg, Germany, September 3-7, 2007.

[Peralta+04] V. Peralta, R. Ruggia, Z. Kedad, M. Bouzeghoub *A Framework for Data Quality Evaluation in a Data Integration System*. 19º Simposio Brasileiro de Banco de Dados (SBBD'2004). Brasil, October 2004.

[Peralta-06] V. Peralta. *Data Quality Evaluation in Data Integration Systems*. PhD Thesis. Université de Versailles, France and Universidad de la República, Uruguay. November, 2006.

[Pon+05] R. K. Pon, A. F. Cárdenas. *Data Quality Inference*. Int. Workshop on Information Quality in Information Systems (IQIS), June 2005, USA.

[Pipino+02] L. L. Pipino, Y. W. Lee, R. Y. Wang. *Data Quality Assessment*. Communications of the ACM. April 2002 / Vol. 45, No. 4ve.

[Quass-99] D. Quass. *A framework for research in data cleaning.* Draft, Brigham Young University, 1999.

[Quix-99] C. Quix. *Repository Support for Data Warehouse Evolution*. DMDW'99

[Rahm+00] E. Rahm, H. H. Do. *Data Cleaning: Problems and Current Approaches.* IEEE Data Engineering Bulletin, Volume 23(4): 3-13 (2000)

[Raman+01] V. Raman, J. Hellerstein. *Potter's Wheel: An Interactive Data Cleaning System*. In Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB'01), Roma, Italy, 2001.

[Rundensteiner+97] E. A. Rundensteiner, A. J. Lee, A. Nica. *On Preserving views in evolving environments.* KRDB'97.

[Scannapieco+02] M. Scannapieco, B. Pernici, E. Pierce. *IP-UML: Towards a Methodology for Quality Improvement Based on the IP-Map Framework.* 7th. International Conference on Information Quality, 2002. (IQ 2002).

[Scannapieco+04] M. Scannapieco, A. Virgillito, C. Marchetti, M. Mecella, R. Baldoni. *The DaQuinCIS Architecture: a Platform for Exchanging and Improving Data Quality in Cooperative Information Systems*. Information Systems 29, 7 (2004), 551–582.

[Scannapieco+05-a] M. Scannapieco, P. Missier, C. Batini. *Data Quality at a Glance*. Datenbank-Spektrum (German data base technology journal, successor of "Datenbank-Rundbrief"), 2005.

[Scannapieco+05-b] M. Scannapieco, B. Pernici, E. M. Pierce. *IP-UML: A Methodology for Quality Improvement based on IP-MAP and UML*. In Advances in Management Information Systems - Information Quality (AMIS-IQ) Monograph, R. Y. Wang, E. M. Pierce, S. E. Madnick, and C. W. Fisher, Eds. Sharpe, M.E., April 2005.

[Shankaranarayan+00] G. Shankaranarayan, R. Wang, M. Ziad. *IP-MAP: Representing the Manufacture of an Information Product.* Proceedings of the 2000 International Conference on Information Quality, October 2000. pp. 1-16.

[Shankaranarayan+03] G. Shankaranarayan, M. Ziad, R. Y. Wang. *Managing Data Quality in Dynamic Decision Making Environments: An Information Product Approach.* Journal of Database Management (14:4), Oct- Dec 2003, pp. 14-32

[Skarra+86] A. H. Skarra, S. B. Zdonik. *The Management of Changing Types in an Object-Oriented Database*. OOP SLA 1986, Portland, Oregon.

[Strong+97] D. M. Strong, Y. W. Lee, R. Y. Wang. *Data Quality in Context*. Communications of the ACM. May 1997/Vol.40, No.5.

[TDQM] http://web.mit.edu/tdqm/www/index.shtml

[Theodoratos+99] D. Theodoratos, M. Bouzeghoub. *Data Currency Quality Factors in Data Warehouse Design.* In Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'99), Germany, 1999.

[Vassiliadis+01] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, T. K. Sellis. *ARKTOS: towards the modeling, design, control and execution of ETL processes.* Information Systems, Volume 26, Number 8: 537-561. 2001.

[Virtamo-04] J. Virtamo. *Poisson Process*. Lecture notes of course Queueing Theory, 2004. Helsinki University of Technology Networking Laboratory.

[Wang+05] R. Y. Wang, K. Chettayar, F. Dravis, J. Funk, R. Katz-Haas, C. Lee, Y. Lee, X. Xian, S. Bhansali. *Exemplifying Business Opportunities for Improving Data Quality from Corporate Household Research.* In Advances in Management Information Systems - Information Quality (AMIS-IQ) Monograph, R. Y. Wang, E. M. Pierce, S. E. Madnick, and C. W. Fisher, Eds. Sharpe, M.E., April 2005.

[Wang+95-a] R. Y. Wang, V. C. Storey, C. P. Firth. *A Framework for Analysis of Data Quality Research.* IEEE Transactions on Knowledge and Data Engineering, 1995. 7(4): p. 623-641.

[Wang+95-b] R. Y. Wang, M. P. Reddy, H. B. Kon. *Toward Quality Data: An Attribute-based Approach.* Decision Support System 13,1995, pp. 349-372.

[Wang-98] R. Y. Wang. *A Product Perspective on Total Data Quality Management.* Communications of the ACM, February 1998, pp. 58-63.

[Weisstein-03] E. W. Weisstein. *Inclusion-Exclusion Principle*. From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Inclusion-ExclusionPrinciple.html. Last accessed on June 2007.

[Wikipedia-07] Wikipedia. URL: www.wikipedia.org. Last accessed on November 2007.

[Zicari-91] R. Zicari. *A Framework for Schema Updates In An Object-Oriented Database System.* GIP Altair, Politecnico di Milano, Milano, Italy, 1991.

APPENDIX I

Preliminary Analysis of the Problem of DIS Quality Changes

Characterization of the phenomenon

In order to study the repercussion of source quality changes on a DIS and with the ultimate goal of proposing some solution for the management of this phenomenon, we start by giving a characterization of the problem.

We have the intuition that this is a new problem which cannot be completely mapped to an existing one, and whose particular features do not allow completely solving it, reusing existing solutions.

In the following we present some characteristics we identify.

Concerning the nature of changes:

- Difficult to predict. Changes in the quality values of a source can be difficult to predict. For example, consider the quality property *response time*. Its value can change at any time, since it depends on factors that are extern to the source, such as the network load. In such cases, statistics can be useful for predicting changes and sometimes avoiding them. If, for example, we know that at a certain time the network traffic will be heavier, perhaps we can change the time of the data transmission from the source, achieving a better performance for the global system.
- Frequent. Changes in source quality can be very frequent, specially when we consider various quality properties, including those that do not depend on the source administrator decisions.
- Not in a certain direction. Quality values may oscillate; a property can change a value and later come back to the same value in several occasions.

Concerning the origin of changes.

- In some cases source quality changes are not decided by the administrator of the source. This may happen in several properties, like *response time*, or *confidence*. Sometimes these changes are caused by changes in the communications or system infrastructure, or by characteristics that the data acquires.
- A quality value of a source may be explicitly changed in order to satisfy a DIS requirement. For example, some procedure at the source level may be improved achieving a better quality value in some property.

Concerning the consequences of changes:

A change in a quality value of a source may have many different consequences, some of them are a direct impact and others are generated by the management of the change. We enumerate them:

- Change in the system quality. In particular, the mediator quality may change and not continue satisfying its required quality.
- Change in the required values of other sources of the system. This may happen in the cases of some quality properties, in which the values of the mediator are a combination of the values of the sources.
- Change in source selection. A source that provided data to the DIS may not continue being of interest to the system, and therefore be left aside.
- Change in the system implementation. The mediator schema as well as the transformation defined for the processing of the data coming from the sources, may be changed in order to recuperate the quality of the system.

Source quality changes vs. source schema evolution

We compare the problem of source quality changes with the problem of source schema evolution in a DIS in order to explore the possible reusing of the existing techniques for managing schema evolution. The following are the differences we found:

- While in schema evolution we find a relatively low frequency of changes, quality changes may have a very high frequency.
- While in schema evolution changes are always applied by the source DBA, quality changes sometimes do not depend on source DBA.
- Schema changes in general are not reverted, while quality changes are sometimes reverted.
- Quality changes statistics can be used for predicting changes, furthermore, probabilistic calculations may be used. In schema evolution, in general, changes cannot be predicted, since they are associated to arbitrary business-related decisions.
- A systematic way for determining the value ranges that must be verified by the source quality values for satisfying the DIS quality requirements, may be defined. In schema evolution it is difficult, or not possible, to define restrictions for the source schemas variations in order to maintain the integrated system stability.
- With respect to the impact of the changes on the sources, in the case of source schema evolution, the impact may very high, since it may affect the rest of the source schema. However, in the case of source quality changes, there is not a direct impact on the source.
- With respect to the impact of the changes on the DIS there is an important difference. Source schema changes automatically generate changes in the mediator schema and in the transformation process (this includes the case where a source or part of it disappears of the system), while source quality changes automatically generate changes only in the mediator quality values. In the case of source quality changes, in order to avoid or minimize mediator quality changes, other modifications may be provoked, such as: mediator schema changes, transformation process changes, elimination of sources, changes on quality values of other sources.

In Table 1 we summarize the presented differences.

	Source schema evolution	**Source quality changes**
Frecuency	Generally Low (not in some cases like Web).	Potentially very High.
Origin	DBA decision.	Not necessarily DBA.
Behavior of changes	Generally not reverted.	May be reverted.
Predictability	Cannot be predicted.	May be predicted.
Valid ranges for changes	Cannot be determined.	May be determined.
Direct impact on the source	Probably high.	No impact.
Direct impact on the DIS	Integrated schema and transformation processes.	DIS quality values.

Table 1: Source schema evolution vs. source quality changes

Despite the differences existing between the two problems, there is an important similarity concerning the management of the source changes. This includes the idea of "change propagation", which means calculating how the source change affects the DIS, following the transformation trace (or mappings generated by the transformation). It also includes the idea of modifying the transformation or generating a change in another source, in order to avoid the change in the integrated schema. Therefore, the similarity consists on the actions that may be carried out for managing the source change.

The characteristics of the problem of source quality changes that differentiate it from the problem of source schema evolution, are the ones that give the possiblities of applying the preventive approach. The similarities between both problems give the possibility of reusing ideas of source schema evolution management, for the management of source quality changes, when we know they will happen or after they have happened.

APPENDIX II

Quality Management Framework - Complete specification

Definition 3.6: The *Quality Management Framework* is a 6-uple QMF = <Sources, Targets, QualityGraph, Properties, Algorithms, ChangeManagementElements>, where.

- Sources is a set of available data sources,
- Targets is a set of data targets,
- QualityGraph is a graph representing the DIS process,
- Properties is a set of functions for describing DISs features and quality measures,
- Algorithms is a set of quality management algorithms,
- ChangeManagementElements are the tools that are used to manage quality changes. □

Definition 3.7: The *ChangeManagementElements* is a 7-uple QMF = <Events, DetectionRules, Statistics, Interpretations, InterpretationRules, Actions, RepairingRules>, where.

- Events is a set of events managed by the QMS,
- DetectionRules is a set of rules that allow detecting relevant quality changes,
- Statistics is a set of functions that give current and historical information about the DIS,
- Interpretations is a set of interpretations about DIS situation,
- InterpretationRules is a set of rules that deduce interpretations from events and statistics,
- Actions is a set of actions that may be applied to the DIS,
- RepairingRules is a set of rules that define, from the interpretations, which actions may be taken for recovering DIS quality □

Definition 3.8: A *data source* is represented by a pair <*Name*, *Description*>, where *Name* is a String that uniquely identifies the source and *Description* is a free-form text providing additional information useful for end-users to identify the source (e.g. URL, provider, high-level content description). □

Definition 3.9: A *data target* is represented by a pair <Name, Description>, where Name is a String that uniquely identifies the data target and Description is a free-form text providing additional information useful for end-users to identify the target (e.g. application/process name, interfaces, servers running the application). □

Definition 3.10: *PropTypes* is a set of Strings, each of which names a property type. A property type is for example, "user quality requirement", "cost", "source quality behavior model". □

Definition 3.11: *PropertiesDefinition* is a function *PD*: PropTypes → U that gives for each property type the domain of the properties of the type, where U is the universe of domains (the set of possible domains for a property). □

Definition 3.12: A *quality graph* is a 5-uple G = (V, E, ρ_V, ρ_E, *gp*) where:

- V is the set of nodes. V^s, V^t and V^a are the sets of source, target and activity nodes respectively; with $V = V^s \cup V^t \cup V^a$. Each source or target node corresponds to a source or target of the framework.
- $E \subset V \times V \times T$ is the set of edges. T ={c, d} distinguishes between control edges (c) and data edges (d). The edge $(u, v)^t$ originates at node *u*, terminates at node *v* and has type *t*; with $u, v \in V$, $t \in T$.
- ρ_V represents the node properties. It is a function that given a node returns a function, which given a property type returns a function, which given a property name returns the value of the property,

  $\rho_V : V \rightarrow \{f / f: \text{PropTypes} \rightarrow (\text{String} \rightarrow \mathcal{U}) \wedge$

  $\forall x \in \text{PropTypes}\ \forall y \in \text{String}\ (f(x))(y) \in PD(x) \cup \{\perp\}\},$

  where $\mathcal{U}$ is the union of the property domains.
- ρ_E represents the edge properties. It is a function that given an edge returns a function, which given a property type returns a function, which given a property name returns the value of the property,

  $\rho_E : E \rightarrow \{f / f: \text{PropTypes} \rightarrow (\text{String} \rightarrow \mathcal{U}) \wedge$

  $\forall x \in \text{PropTypes}\ \forall y \in \text{String}\ (f(x))(y) \in PD(x) \cup \{\perp\}\},$

  where $\mathcal{U}$ is the union of the property domains.
- *gp* represents the graph properties. It is a function that for each property type, gives a function that for each property name gives the value of the property,

  gp: $\text{PropTypes} \rightarrow \{f / f: \text{String} \rightarrow \mathcal{U} \wedge$

  $\forall x \in \text{PropTypes}, \forall y \in \text{String}, (gp(x))(y) \in PD(x) \cup \{\perp\}\},$

  where $\mathcal{U}$ is the union of the property domains. □

Definition 3.13: *User Quality Requirement* is a property type, whose corresponding domain is a set of 4-uples of the form: <qfactor, type, value, prob>, where:

- qfactor is a String, representing the quality factor.
- type ∈ {"probability", "maximum", "minimum", "average", "frequency"), tells the type of the requirement.
- value is a Decimal, the quality value of the requirement.
- prob ∈ 0..1, is the probabilistic value associated to the quality value. Used only when type = "probability". □

Definition 4.1: A *source quality behavior model* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, source, distribution>, where

- qfactor is a String, representing the quality factor,
- source is a String, the name of the source to which the model corresponds,
- distribution is a set of pairs <qvalue, probability>, where
  - qvalue is a Decimal number and
  - probability is a number between 0 and 1, representing the probability of the quality value.

□

Definition 4.2: A *source restriction* is a restriction to be verified by the quality factor of the source. It is a 4-uple r = <qfactor, source, op, value>, where:

- qfactor is a String, representing the quality factor
- source is a source node of the quality graph
- op ∈ {"<", "≤", ">", "≥", "="}, is a comparison operator
- value is a Decimal □

Definition 4.3: A *restriction vector* is a set of source restrictions, one restriction for each one of the DIS sources, where all the restriction operators are the same. We use the following notation:

$v = < r_{S1}, \ldots, r_{Sn}>$, where r_{Si} is the restriction associated to source S_i □

Definition 4.4: A *restriction-vector space* is a set of restriction vectors. We use the following notation:

$\mathcal{V} = \{v_1, \ldots, v_m\}$, where v_i is a restriction vector. □

Definition 4.5: The *accepted configurations* is a property type, whose corresponding domain is a set of pairs of the form: <req-set, rv-space>, where:

- reqs-set is a set of pairs of the form <target, req-name>, where:
  - target is the data target to which the requirement is associated
  - req-name is a String (the name of the requirement)
- rv-space is a restriction-vector space, which corresponds to the requirements of reqs-set. □

Definition 4.8: A *quality-values vector* is a set of quality values, containing one value for each one of the DIS sources. We use the following notation:

$vv = < qv_{S1}, \ldots, qv_{Sn}>$, where qv_{Si} is the quality value associated to source S_i □

Definition 4.9: A *DIS quality satisfaction model* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, requirements-set, sat-probability>, where

- qfactor is a String, representing the quality factor,
- requirements-set is a set of Requirements,
- sat-probability is a number between 0 and 1, representing the probability of the satisfaction of these requirements by the DIS. □

Definition 4.10: A *DIS quality distribution model* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, data-target, distribution>, where

- qfactor is a String, representing the quality factor,
- data-target is a String, the name of the data target to which the model corresponds,
- distribution is a set of pairs <qvalue, probability>, where
    - qvalue is a Decimal number and
    - probability is a number between 0 and 1, representing the probability of the quality value.

□

Definition 4.11: A *source quality models history* is a property type, whose corresponding domain is a set of 3-uples of the form: <qfactor, source, distributions>, where

- qfactor is a String, representing the quality factor,
- source is a String, the name of the source to which the model corresponds,
- distributions is a set of 5-uples, indicators = <minimum, maximum, expectation, mode, timestamp>, where
    - minimum, maximum, expectation and mode are Decimal numbers,
    - timestamp is a date/time field, representing the instant when the distribution was stated as current. □

Definition 6.1 (from [Peralta-06]): A *data path* in a quality graph is a sequence of nodes of the graph, where each node is connected to its successor in the sequence by a data edge. We denote a data path, giving the sequence of nodes that compose it, comma separated and between square brackets, for example $[A_0,A_1,A_3,A_4]$. We also use suspension points for omitting intermediate nodes, for example $[A_0,\ldots A_4]$. □

Definition 6.2 (from [Peralta-06][1]): Given a data path in a quality graph $[A_0,A_1,\ldots A_p]$, starting at a source node A_0, the *path freshness* is the freshness value propagated along the path (ignoring other nodes of the graph), i.e. it is the sum of source data freshness of the source node, the processing costs of the nodes in the path and the inter-process delays between the nodes:

$$PathFreshness([A_0,\ldots A_p]) = SourceFreshness(A_0) + \sum_{x=0..p} Cost(A_x) + \sum_{x=1..p} InterProcessDelay(A_{x-1},A_x)$$ □

InterProcessDelay is the time that necessarily passes between two successive activities, without considering synchronization delays. That is to say it may exist a time interval between two activities generated by the way one activity passes data to the other, the frequency, etc.

Definition 6.3 (from [Peralta-06] [2]): Given an activity node A_p, a *critical path* for A_p is a data path $[A_0,\ldots A_p]$, from a source node A_0, for which the freshness of data produced by node A_p (delivered to each successor) equals the path freshness.

$$Freshness(A_p) = PathFreshness([A_0,\ldots A_p])$$

[1] With minimum nomenclature modifications.

[2] With minimum nomenclature modifications.

Given a target node T_i, a critical path for T_i is the critical path of its predecessor activity. □

APPENDIX III

Change Detection Rules - Complete specification

Events

ClassName	Attribute	Attribute Description
QModelChange	Qgraph	Quality Graph where the change occurred.
	Qfactor	Quality factor corresponding to the changed model.
	SourceName	Name of the source whose quality model has changed.
	ChangedIndicators	Set of model indicators (min, expectation, etc.) that changed.
	Timestamp	Date-time of the change.
TGraphChange	QGraph	Quality Graph where the change occurred.
	QFactor	Quality factor affected by the change.
	Type	Type of change that has occurred on the transformation graph. Some possible values: "structure", "activity-cost", "activity-effectiveness".
	ActivityName	Only in the case of change on certain graph activity.
	Timestamp	Date-time of the change.
QReqChange	QGraph	Quality Graph where the change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the change.
QSatisfactionChange	QGraph	Quality Graph where the change occurred.
	Requirements	Considered requirements.
	OriginalChange	Change that generated the level-1 event.
	OriginalSource	Source where the change occurred.
	GraphChange	Kind of change that occurred on the transformation graph.
	ChangedActivity	Graph activity where a change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the original change.
CertaintyChange	QGraph	Quality Graph where the change occurred.
	QFactor	Quality factor affected by the change.
	OriginalChange	Change that generated the level-1 event.

	OriginalSource	Source where the change occurred
	GraphChange	Kind of change that occurred on the transformation graph.
	ChangedActivity	Graph activity where a change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the original change.
QValuesDissatisfaction	QGraph	Quality Graph where the change occurred.
	Requirements	Set of requirements that are not satisfied.
	OriginalChange	Change that generated the level-1 event.
	OriginalSource	Source where the change occurred.
	GraphChange	Kind of change that occurred on the transformation graph.
	ChangedActivity	Graph activity where a change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the original change.
ProbabilityDissatisfaction	QGraph	Quality Graph where the change occurred.
	QFactor	Quality factor.
	OriginalChange	Change that generated the level-1 event.
	OriginalSource	Source where the change occurred.
	GraphChange	Kind of change that occurred on the transformation graph.
	ChangedActivity	Graph activity where a change occurred.
	ChangedReq	Requirement that has changed.
	Timestamp	Date-time of the original change.

Change Detection Rules

Rules for *QModelChange* events:

EVENT: e1: *QModelChange*
CONDITION: SelectReqs (‘minimum’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName)) $\neq \varnothing$
AND ‘minimum’ $\in$ e1.ChangedIndicators
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = SelectReqs (‘minimum’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName))
- OriginalChange = ‘QModelChange’
- OriginalSource = e1.SourceName
- GraphChange = NULL

- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *QModelChange*
CONDITION: SelectReqs (‘maximum’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName)) ≠ ∅
AND ‘maximum’ ∈ e1.ChangedIndicators
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = SelectReqs (‘maximum’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName))
- OriginalChange = ‘QModelChange’
- OriginalSource = e1.SourceName
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *QModelChange*
CONDITION: SelectReqs (‘average’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName)) ≠ ∅
AND ‘expectation’ ∈ e1.ChangedIndicators
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = SelectReqs (‘average’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName))
- OriginalChange = ‘QModelChange’
- OriginalSource = e1.SourceName
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *QModelChange*
CONDITION: SelectReqs (‘frequency’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName)) ≠ ∅
AND ‘mode’ ∈ e1.ChangedIndicators
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = SelectReqs (‘frequency’, GetCorrespondingRequirements (e1.QGraph, e1.QFactor, e1.SourceName))
- OriginalChange = ‘QModelChange’
- OriginalSource = e1.SourceName
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *QModelChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, ‘probability’) ≠ ∅
ACTION: Create event e2: *CertaintyChange*, attribute values:

- QGraph = e1.QGraph
- QFactor = e1.QFactor
- OriginalChange = 'QModelChange'
- OriginalSource = e1.SourceName
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = NULL
- Timestamp = e1.Timestamp

Rules for *TGraphChange* events:

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, 'minimum') ≠ ∅
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, e1.QFactor, 'minimum')
- OriginalChange = 'TGraphChange'
- OriginalSource = NULL
- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, 'maximum') ≠ ∅
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, e1.QFactor, 'maximum')
- OriginalChange = 'TGraphChange'
- OriginalSource = NULL
- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, 'average') ≠ ∅
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, e1.QFactor, 'average')
- OriginalChange = 'TGraphChange'
- OriginalSource = NULL
- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, 'frequency') ≠ ∅
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, e1.QFactor, 'frequency')

- OriginalChange = ‘TGraphChange’
- OriginalSource = NULL
- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

EVENT: e1: *TGraphChange*
CONDITION: Get_Requirements (e1.QGraph, e1.QFactor, ‘probability’) ≠ ∅
ACTION: Create event e2: *CertaintyChange*, attribute values:
- QGraph = e1.QGraph
- QFactor = e1.QFactor
- OriginalChange = ‘TGraphChange’
- OriginalSource = NULL
- GraphChange = e1.Type
- ChangedActivity = e1.ActivityName
- ChangedReq = NULL
- Timestamp = e1.Timestamp

Rules for *QReqChange* events:

EVENT: e1: *QReqChange*
CONDITION: GetType(e1.ChangedReq) = ‘minimum’
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, GetQFactor(e1.ChangedReq), ‘minimum’)
- OriginalChange = ‘QReqChange’
- OriginalSource = NULL
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

EVENT: e1: *QReqChange*
CONDITION: GetType(e1.ChangedReq) = ‘maximum’
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, GetQFactor(e1.ChangedReq), ‘maximum’)
- OriginalChange = ‘QReqChange’
- OriginalSource = NULL
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

EVENT: e1: *QReqChange*
CONDITION: GetType(e1.ChangedReq) = ‘average’
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph,

GetQFactor(e1.ChangedReq), 'average')
- OriginalChange = 'QReqChange'
- OriginalSource = NULL
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

EVENT: e1: *QReqChange*
CONDITION: GetType(e1.ChangedReq) = 'frequency'
ACTION: Create event e2: *QSatisfactionChange*, attribute values:
- QGraph = e1.QGraph
- Requirements = Get_Requirements (e1.QGraph, GetQFactor(e1.ChangedReq), 'frequency')
- OriginalChange = 'QReqChange'
- OriginalSource = NULL
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

EVENT: e1: *QReqChange*
CONDITION: GetType(e1.ChangedReq) = 'probability'
ACTION: Create event e2: *CertaintyChange*, attribute values:
- QGraph = e1.QGraph
- QFactor = GetQFactor(e1.ChangedReq)
- OriginalChange = 'QReqChange'
- OriginalSource = NULL
- GraphChange = NULL
- ChangedActivity = NULL
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

Rule for *QSatisfactionChange* events:

EVENT: e1: *QSatisfactionChange*
CONDITION: NOT QualitySatisfaction (e1.QGraph,
Accepted_Conf_Calculation (e1.QGraph,
GetQFactor(e1.Requirements), e1.Requirements),
GetQFactor(e1.Requirements),
GetType(e1.Requirements))
ACTION: Create event e2: *QValuesDissatisfaction*, attribute values:
- QGraph = e1.QGraph
- Requirements = e1.Requirements
- OriginalChange = e1.OriginalChange
- OriginalSource = e1.OriginalSource
- GraphChange = e1.GraphChange
- ChangedActivity = e1.ChangedActivity
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

Rule for *CertaintyChange* events:

EVENT: e1: *CertaintyChange*
CONDITION: NOT Quality_Certainty_Verification (e1.QGraph, e1.QFactor)
ACTION: Create event e2: *ProbabilityDissatisfaction*, attribute values:
- QGraph = e1.QGraph
- QFactor = e1.QFactor
- OriginalChange = e1.OriginalChange
- OriginalSource = e1.OriginalSource
- GraphChange = e1.GraphChange
- ChangedActivity = e1.ChangedActivity
- ChangedReq = e1.ChangedReq
- Timestamp = e1.Timestamp

Description of used auxiliary functions:

Function **SelectReqs** selects from a set of user quality requirements, those whose type is the given one.

Function **GetCorrespondingRequirements** obtains for a source, the quality requirements of certain quality factor, that involve it.

Function **Quality_Certainty_Verification** receives a quality graph and a quality factor, and verifies if the DIS Quality Certainty satisfies the user quality requirements (specified in Chapter 5, Section 2).

Function **Get_Requirements** obtains, given a quality graph and a quality factor, all the requirements of certain type (which are associated to data targets of the given graph).

Function **Accepted_Conf_Calculation** was specified in Chapter 4, Section 4.1. It calculates the accepted configurations for a quality graph, a set of requirements and a quality factor.

Function **QualitySatisfaction** was specified in Chapter 4, Section 4.3. Given a requirement type and the accepted configurations for a set of requirements of this type, this function determines if the sources satisfy the accepted configurations.

Function **GetType** returns the type of a user quality requirement. We overload this function, so that it can receive one requirement or a set of requirements. In the second case it returns the type of any of the requirements.

Function **GetQFactor** returns the quality factor of a user quality requirement. We overload this function, so that it can receive one requirement or a set of requirements. In the second case it returns the quality factor of any of the requirements.