Data Quality Evaluation in Data Integration Systems

Veronika Peralta

Université de Versailles Saint-Quentin-en-Yvelines (France) and
Universidad de la República (Uruguay)

Data Quality Evaluation in Data Integration Systems

by

Verónika PERALTA

PhD Thesis

for obtaining the degree of **Doctor of Philosophy**
in **Computer Science**

Versailles, November 17th, 2006

Committee

Jacky AKOKA	Professor, CNAM, Paris (reviewer)
Catherine BERRUT	Professor, IMAG, Grenoble
Mokrane BOUZEGHOUB	Professor, Université de Versailles (advisor)
Sylvie CALABRETTO	Assistant Professor, INSA, Lyon (reviewer)
José COCH	Director of Services and Support, LINGWAY, Paris
Nicole LEVY	Professor, Université de Versailles (chair)
Raúl RUGGIA	Professor, Universidad de la República, Uruguay (advisor)

Acknowledges

I extend my sincere gratitude and appreciation to many people who made this PhD thesis possible. The first persons I would like to thank are my supervisors Mokrane Bouzeghoub and Raúl Ruggia, who supported me and placed their trust in me. I owe you lots of gratitude for having taught me so much during these years. *Merci Mokrane ! ¡Gracias Raúl!*

I would like to thank Professors Jacky Akoka, Catherine Berrut, Sylvie Calabretto, José Coch and Nicole Levy for having accepted to be members of my PhD committee. You monitored my work and took effort in reading and providing me with valuable comments.

I also owe a lot of thanks to all my colleges of the InCo Laboratory at the University of the Republic of Uruguay, who introduced me to the fabulous world of research during my master, encouraged me to start PhD and gave me great advices all along my career. Special thanks to Adriana Marotta, with whom I shared this quest from the beginning. I really enjoy working with you and I wish you a lot of success for your PhD dissertation. Many thanks to Regina Motz for sharing so many adventures with me. Many thanks to Martín Barrere, Fernando Carpani, Lorena Etcheverry, Pablo Gatto, Joaquín Goyoaga, Alejandro Gutiérrez, Jacqueline Guzmán, Ignacio Larrañaga, Federico Piedrabuena, Lydia Silva, Salvador Tercia and Diego Vallespir for being so nice friends and giving me the feeling of being at home at work. I would also like to thank Raquel Sosa, Enrique Delfino, Pablo Alzuri and María Freira for being fantastic friends and working late with me so many times. Very special thanks to Alvaro Illarze for his unconditional friendship and for the numerous times he helped me with my English. *¡Muchas gracias a todos!*

I would also like to thank all my colleagues of the PRiSM Laboratory at the University of Versailles St-Quentin-en-Yvelines, who received me and gave me the feeling of being in my country. Special thanks to Xiaohui Xue and Tao Wan, my Chinese sisters, who started and finished PhD with me. It was very nice to share so many hours with you during my studies; I hope we find new challenges to try together. I would also like to thank Dimitre Kostadknov and Juan Carlos Corrales. It was nice to work with you in a friendly environment; good luck for your PhD dissertations! Many thanks to Zoubida Kedad, who always was there for giving her sincere advice. Many thanks to Daniela Grigori, Stéphane Lopes and Assia Soukane for bringing a cheerful atmosphere at work. I would also like to thank Armando Borrero, Michel Cavaillé, François Dang-Ngoc, Tuyet-Trâm Dang-Ngoc, Sébastien Donadio, Clément Jamard, Huaizhong Kou, Octavio Ramírez, Romain Ravaux, Nicolas Travers, Daniel Vila Monteiro and Lilan Wu for being fantastic colleagues and friends and for the numerous times they helped me with my French. Special thanks to David Nott for his invaluable encouragement, patience and sense of humor. *Merci à vous tous !*

I must thank Lucyla Alonso, Annick Baffert, Laura Bermúdez, Joseline Cortazo, Chantal Ducoin, Isabelle Kretz, Catherine Le Quere, Isabelle Moudenner, Martiniano Olivera, Lucía Píriz and Mabel Seroubian, who helped me many times on administrative issues. I would also like to thank Julián Adib, Alejandro Blanco, Juan Diego Ferré, Jean-Louis Jammier, Marcelo Rodríguez, Alexander Sklar, Jorge Sotuyo and Felipe Zipitría for their useful support in networking issues.

I also thanks some people I met during my studies: Stéphane Boll, Daniel Calegari, Cosmin Cremarenco, Frédéric Dang-Ngoc, Mathieu Decore, Nicolas Dieu, Sophie Giraud, Aurelian Lavric, Cristian Saita, Christophe Salperwyck and Fei Sha for their kind friendship. Special thanks to Abdelkrim Lahlou, who taught me how to live in France and always offered me his friendly support. Many thanks to Noel Vidart and Gustavo Betarte for being my adoptive parents in France. *Merci pour votre amitié !*

I also owe a lot of thanks to my friends Eduardo Assandri, Santiago Fraschini, Luis Lena, Mario Maneyro, Julio Russo and Karina Talasimov, who accompanied me since my first day at University. Thank you for your encouragement and friendship. Many thanks to Beatriz Castro, Viviana García, Diego Olave, Marcos Orfila, Andrea Pereira, Mariela Questa, Leonardo Renard, Andrea Rodríguez and Rossanna Wirth for being so fabulous friends and for encouraging me, even by mail, all along this adventure. *¡Gracias por ser tan buenos amigos!*

Finally, I owe a lot of thanks to my family, who always supported me along my life: my parents, my sister Ana Laura and his husband Diego, my little sister Camila, my aunts Elbia, Esther, Maruja and Virmar, my cousins Andreia, Gustavo, Joaquín, Jorge, Juan Manuel, Pilar, Silvia and Valeria. *¡Los quiero muchísimo!*

Abstract

The needs of accessing in a uniform way to information available in multiple data sources are increasingly higher and generalized, particularly in the context of decision making applications which need a comprehensive analysis and exploration of data. With the development of Data Integration Systems (DIS), information quality is becoming a *first class* property which is more and more required by end-users.

This thesis deals with data quality evaluation in DIS. Specifically, we address the problems of evaluating the quality of the data conveyed to users in response to their queries and verifying if users' quality expectations can be achieved. We also analyze how quality measures can be used for improving the DIS and enforcing data quality. Our approach consists in studying one quality factor at a time, analyzing its impact within a DIS, proposing techniques for its evaluation and proposing improvement actions for its enforcement. Among the quality factors that have been proposed, this thesis analyzes two main ones: *data freshness* and *data accuracy*.

We analyze the different definitions and metrics proposed for data freshness and data accuracy and we abstract the properties of the DIS that impact on their evaluation. We summarize the analysis of each factor with a taxonomy, which allows comparing existent works and highlighting open problems.

We propose a quality evaluation framework that models the different elements involved in data quality evaluation, namely: data sources, user queries, DIS processes, DIS properties, quality measures and quality evaluation algorithms. In particular, DIS processes are modeled as workflow processes in which the workflow activities perform the different tasks that extract, integrate and convey data to end-users. Our reasoning support for quality evaluation is a direct acyclic graph, called quality graph, which has the same workflow structure than the DIS and contains, as labels, the DIS properties that are relevant for quality evaluation. We develop quality evaluation algorithms that take as input source data quality values and DIS property values and combine such values obtaining a value for the data conveyed by the DIS. They are based on the graph representation and combine property values while traversing the graph. Evaluation algorithms can be instantiated for taking into account the properties that influence data quality in a particular application. The idea behind the framework is to define a flexible context which allows specializing evaluation algorithms for specific application scenarios.

The quality values obtained during data quality evaluation are compared to those expected by users. If quality expectations are not satisfied, several improvement actions can be taken. We suggest some elementary improvement actions that can be composed to improve data quality in concrete DISs. For enforcing data freshness, we propose the analysis of the DIS at different abstraction levels in order to identify its critical points (the portions of the DIS that cause the non-achievement of freshness expectations) and to target the study of improvement actions for these points. For enforcing data accuracy, we propose the partitioning of query result in areas (some attributes, some tuples) having homogeneous accuracy. This allows user applications to retrieve only the most accurate data, to filter data not satisfying an accuracy threshold or to incrementally convey areas (e.g. displaying first the most accurate areas). Our approach differentiates from existing source selection proposals because we allow the selection of the areas having the best accuracy instead of only selecting whole relations.

The main contributions of this thesis are: (i) a detailed analysis of data freshness and data accuracy quality factors; (ii) the proposal of techniques and algorithms for the evaluation and enforcement of data freshness and data accuracy; and (iii) a prototype of a quality evaluation tool oriented to be used in practical contexts of DIS management.

Résumé

Les besoins d'accéder, de façon uniforme, à des sources de données multiples, sont chaque jour plus forts, particulièrement, dans les systèmes décisionnels qui ont besoin d'une analyse compréhensive des données. Avec le développement des Systèmes d'Intégration de Données (SID), la qualité de l'information est devenue une propriété de premier niveau de plus en plus exigée par les utilisateurs.

Cette thèse porte sur la qualité des données dans les SID. Nous nous intéressons, plus précisément, aux problèmes de l'évaluation de la qualité des données délivrées aux utilisateurs en réponse à leurs requêtes et de la satisfaction des exigences des utilisateurs en terme de qualité. Nous analysons également l'utilisation de mesures de qualité pour l'amélioration de la conception du SID et de la qualité des données. Notre approche consiste à étudier un facteur de qualité à la fois, en analysant sa relation avec le SID, en proposant des techniques pour son évaluation et en proposant des actions pour son amélioration. Parmi les facteurs de qualité qui ont été proposés, cette thèse analyse deux facteurs de qualité : *la fraîcheur* et *l'exactitude* des données.

Nous analysons les différentes définitions et mesures qui ont été proposées pour la fraîcheur et l'exactitude des données et nous faisons émerger les propriétés du SID qui ont un impact important sur leur évaluation. Nous résumons l'analyse de chaque facteur par le biais d'une taxonomie, qui sert à comparer les travaux existants et à faire ressortir les problèmes ouverts.

Nous proposons un canevas qui modélise les différents éléments liés à l'évaluation de la qualité tels que les sources de données, les requêtes utilisateur, les processus d'intégration du SID, les propriétés du SID, les mesures de qualité et les algorithmes d'évaluation de la qualité. En particulier, nous modélisons les processus d'intégration du SID comme des processus de workflow, dans lesquels les activités réalisent les tâches qui extraient, intègrent et envoient des données aux utilisateurs. Notre support de raisonnement pour l'évaluation de la qualité est un graphe acyclique dirigé, appelé graphe de qualité, qui a la même structure du SID et contient, comme étiquettes, les propriétés du SID qui sont relevants pour l'évaluation de la qualité. Nous développons des algorithmes d'évaluation qui prennent en entrée les valeurs de qualité des données sources et les propriétés du SID, et, combinent ces valeurs pour qualifier les données délivrées par le SID. Ils se basent sur la représentation en forme de graphe et combinent les valeurs des propriétés en traversant le graphe. Les algorithmes d'évaluation peuvent être spécialisés pour tenir compte des propriétés qui influent la qualité dans une application concrète. L'idée derrière le canevas est de définir un contexte flexible qui permet la spécialisation des algorithmes d'évaluation à des scénarios d'application spécifiques.

Les valeurs de qualité obtenues pendant l'évaluation sont comparées à celles attendues par les utilisateurs. Des actions d'amélioration peuvent se réaliser si les exigences de qualité ne sont pas satisfaites. Nous suggérons des actions d'amélioration élémentaires qui peuvent être composées pour améliorer la qualité dans un SID concret. Notre approche pour améliorer la fraîcheur des données consiste à l'analyse du SID à différents niveaux d'abstraction, de façon à identifier ses points critiques et cibler l'application d'actions d'amélioration sur ces points-là. Notre approche pour améliorer l'exactitude des données consiste à partitionner les résultats des requêtes en portions (certains attributs, certaines tuples) ayant une exactitude homogène. Cela permet aux applications utilisateur de visualiser seulement les données les plus exactes, de filtrer les données ne satisfaisant pas les exigences d'exactitude ou de visualiser les données par tranche selon leur exactitude. Comparée aux approches existantes de sélection de sources, notre proposition permet de sélectionner les portions les plus exactes au lieu de filtrer des sources entières.

Les contributions principales de cette thèse sont : (1) une analyse détaillée des facteurs de qualité fraîcheur et exactitude ; (2) la proposition de techniques et algorithmes pour l'évaluation et l'amélioration de la fraîcheur et l'exactitude des données ; et (3) un prototype d'évaluation de la qualité utilisable dans la conception de SID.

Resumen

La necesidad de acceder de manera uniforme a múltiples fuentes de datos es cada día más fuerte y generalizada, especialmente en el contexto de aplicaciones para toma de decisiones, las cuales necesitan de un análisis comprensivo y exploratorio de los datos. Con el desarrollo de Sistemas de Integración de Datos (SID), la calidad de la información se ha transformado en una propiedad de primer nivel, cada vez más requerida por los usuarios.

Esta tesis trata sobre la evaluación de la calidad de los datos en los SID. En particular, se abordan los problemas de la evaluación de la calidad de los datos que responden a consultas de usuarios y la satisfacción de las exigencias de dichos usuarios en términos de calidad. Se analiza también la utilización de medidas de calidad para mejorar el diseño del SID e incrementar la calidad de los datos. Nuestro enfoque consiste en estudiar un factor de calidad a la vez, analizando su impacto en el SID, proponiendo técnicas para su evaluación y proponiendo acciones para su mejora. Entre los factores de calidad que se han propuesto en la literatura, esta tesis analiza dos de los más usados: *la frescura* y *la exactitud* de los datos.

Analizamos las diferentes definiciones y métricas que se han propuesto para la frescura y la exactitud de los datos y abstraemos las propiedades del SID que juegan un rol importante en su evaluación. El análisis de cada factor se resume en una taxonomía, la cual permite comparar los trabajos existentes y resaltar los problemas abiertos.

Proponemos un marco de trabajo que modela los diferentes elementos relacionados a la evaluación de la calidad: fuentes de datos, consultas de usuarios, procesos de integración del SID, propiedades del SID, medidas de calidad y algoritmos de evaluación de la calidad. En particular, los procesos de integración del SID se modelan como flujos de trabajo, cuyas actividades realizan las tareas de extracción, integración y entrega de los datos a los usuarios. Nuestro soporte de razonamiento para la evaluación de la calidad es un grafo acíclico dirigido, llamado grafo de calidad, que tiene la misma estructura del SID y está etiquetado con las propiedades del SID que son relevantes para la evaluación de la calidad. Los algoritmos de evaluación de la calidad toman como entrada los valores de calidad de los datos fuentes y las propiedades del SID y combinan dichos valores obteniendo una medida de la calidad de los datos retornados por el SID. Los algoritmos se basan en la representación de grafo y combinan los valores de las propiedades mientras recorren el grafo. Los mismos pueden instanciarse para tener en cuenta las propiedades que influencian la calidad en una aplicación concreta. La idea detrás del marco de trabajo es definir un contexto flexible que permita la especialización de los algoritmos para escenarios de aplicación específicos.

Los valores de calidad obtenidos durante la evaluación se comparan con los valores exigidos por los usuarios. Si las exigencias de calidad no son satisfechas, se pueden realizar acciones de mejora al SID. Sugerimos un conjunto básico de acciones de mejora que pueden componerse para mejorar la calidad en SID concretos. Para mejorar la frescura de los datos proponemos analizar el SID a diferentes niveles de abstracción, de manera de identificar sus puntos críticos (las porciones del SID que causan la no satisfacción de las exigencias de frescura) y concentrar la aplicación de acciones de mejora sobre esos puntos. Para mejorar la exactitud de los datos proponemos partir los resultados de las consultas en áreas (algunos atributos, algunas tuplas) de exactitud homogénea. Esto permite que las aplicaciones de los usuarios desplieguen solamente los datos más exactos, filtren los datos que no satisfacen las exigencias de exactitud o desplieguen los datos en camadas según sus exactitudes. Nuestro enfoque se diferencia de los enfoques existentes de selección de fuentes porque podemos seleccionar áreas de buena exactitud en lugar de sólo seleccionar fuentes enteras.

Las principales contribuciones de esta tesis son: (i) un análisis detallado de los factores de calidad frescura y exactitud, (ii) la propuesta de técnicas y algoritmos de evaluación y mejora de la frescura y la exactitud de los datos, y (iii) un prototipo de evaluación de la calidad utilizable en contextos prácticos de diseño de SID.

Content

Chapter 1. Introduction

This chapter introduces data quality evaluation in data integration systems, describes the addressed problems and presents an overview of our proposal for solving such problems.

1. Context

The technological advances of the last years allowed the development of information systems of wide scope, which offer access to large volumes of information distributed in multiple heterogeneous data sources. Although these systems have been proposed and used for more than a decade, they became more important in the last years, both at academic and industrial level. The increasing interest in this type of systems is mainly due to the proliferation of data available in remote sites, frequently stored in diverse platforms and with diverse formats. In particular, the World Wide Web has become a major source of information about all areas of interest, which can be used by information systems as any data supplier.

The needs of accessing in a uniform way to information available in multiple data sources are increasingly higher and generalized, particularly in the context of decision making applications which need a comprehensive analysis and exploration of data. The *Data Integration Systems* (DISs) appeared in response to these needs. A DIS is an information system that integrates data of different independent data sources and provides the vision of a unique database to users. Users pose queries via an access interface and the DIS answers their queries with information obtained and synthesized from source data. Information integration is the problem of combining the data residing at different sources and providing the user with a unified schema of these data, called global schema [Calvanese+2001], which represents the potential requirements of users. The global schema is therefore a reconciled view of the information, which can be queried by the user. It can be thought as a set of relations, potentially virtual (in the sense that their extensions may not be actually stored anywhere). A data integration system liberates the user from having to locate the sources relevant to a query, interact with each source in isolation, and manually combine the data from different sources.

Examples of DISs are *Mediation Systems* [Wiederhold 1992], which provide access to data extracted from several sources and integrated in a transparent way in response to user queries. *Data Warehousing Systems* [Inmon 1996], also extract, transform and integrate data from various, possibly heterogeneous, sources, aggregate and materialize information from this data and make it available for strategic analysis to the decision makers. Other examples of DISs are *Web Portals,* which provide access to subject-oriented information acquired and synthesized from Web sources, generally caching important amounts of data [Bright+2002].

In a context of DISs providing access to large amounts of data from alternative sources and conveying alternative query answers to users, information quality is becoming a *first class* property increasingly required by end-users. As the potentially retrieved data grows, users are more concerned about data quality [Wang+1996] [Gertz+2004] [Ballou+1998]. Some surveys and empirical studies have showed the importance of data quality for end users, in particular, when dealing with heterogeneous data coming from distributed autonomous sources [Wang+1996] [Mannino+2004] [Shin 2003].

Assuring the quality of the data conveyed to users is an important problem, which is closely related to the success of information systems. Numerous works in the areas of Information Systems and Software Engineering deal with quality control and quality assurance [Ballou+1998] [Pipino+2002] [Bobrowski+1998]. In the case of DISs, the problem is particularly complex due to the integration of data coming from multiple sources and possibly having different quality. Because of the great number and high diversity of data sources as well as their autonomy, it is important to have a fine knowledge of their quality and to take it into account during DIS design. In addition, information quality problems have been reported as critical in several scientific and social areas such as Environment [Jankowka 2000] [USEPA 2004], Genetics [Müller+2003a], Economy [Mazzi+2005] and Informatics in the Web [Gertz+2004]. Solving data quality problems opens a door to consider data production as any other item production.

2. Motivations and problems

Data quality evaluation in DISs consists in calculating the quality of the data conveyed to the users. Generally, data quality is best described or characterized via multiple attributes or factors, which describe certain properties about the data delivered to users (e.g. freshness, accuracy, completeness) and about the processes that manipulate this data (e.g. response time, reliability, security). Consequently, data quality evaluation consists in calculating several quality attributes, each one describing a specific quality aspect of data.

The quality of the information conveyed to users depends on the internal quality of source data (coherence, completeness, freshness, etc.) and on the characteristics of the processes that extract and integrate such data (policies, constraints, costs, delays, implementation features, etc.).

For example, let us consider a user asking for the most popular children videos and demanding certain levels of completeness, accuracy, freshness and confidence for query result. The quality of the delivered data depends on the intrinsic data sources and on the way of integrating data (e.g. the execution cost of the integration process may influence data freshness). As there may be several sources providing the same type of data, several extraction and integration processes can be conceived for conveying data to users. For example, different processes can provide children popular videos, e.g.: (P1) returning data from Disney®, (P2) returning data from Amazon®, (P3) integrating data from Amazon® and Block-Buster®, and (P4) conveying data from Film-Critiquer®, as illustrated in Figure 1.1.

Two problems arise: (i) deciding which sources to query and (ii) deciding how to merge the obtained data. These decisions can be taken considering the quality of data delivered by each process. In addition, the perception on data quality depends on users expectations, for example, some users do not care if the video list is quite incomplete if they have a rapid and accurate response; conversely, other users may want to examine all videos, even if they must wait additional time.

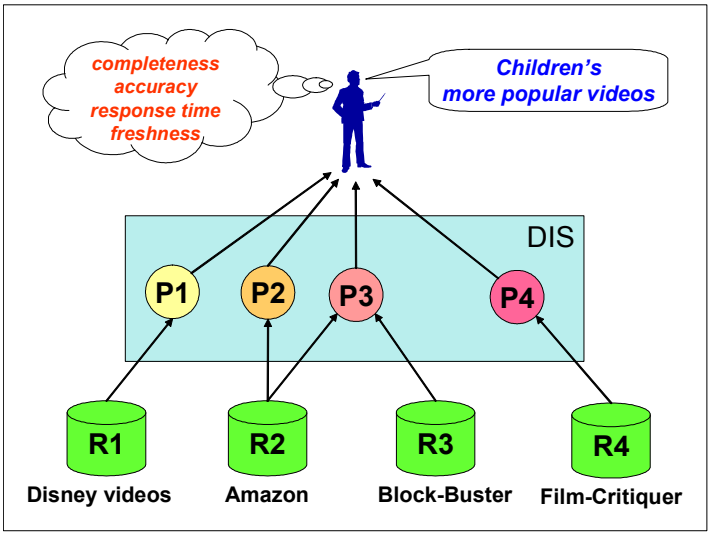

Figure 1.1 – Alternative responses to a DIS query

Consequently, in order to evaluate the quality of data conveyed to users we should study the quality of source data and the characteristics of the DIS integration processes. We should also analyze how to combine all these elements in order to aggregate a value that qualifies the delivered data.

In summary, data quality evaluation in DISs involves a certain number of technical issues:

- *Analysis of quality factors and metrics*: This concerns the analysis of quality factors, the definition of appropriate metrics for measuring them and the study of the DIS properties that have impact in their evaluation (for example, the execution delay of the system may impact data freshness).
- *Definition of user quality expectations*: This concerns the study of the quality factors that are more appropriate for representing user quality needs and the specification of user quality expectations according to such quality factors and metrics.

- *Assessment of source data quality*: This concerns the specification and implementation of measurement procedures for measuring the quality of source data (or a sample of source data) according to the analyzed quality factors and metrics.
- *Assessment of DIS property values*: This concerns the specification and implementation of measurement processes for taking measures or estimating DIS property values (for example, routines for measuring the average response time of the DIS).
- *Quality auditing*: Having measures of source data quality and DIS properties, such measures should be aggregated into a value that qualifies the delivered data. This concerns the study of the influence of those properties in specific quality factors and the specification of aggregation functions. The confrontation of the calculated quality values with those expected by users allows deciding if user quality expectations can be achieved and in which degree.
- *Quality improvement*: This concerns the specification of improvement actions for enforcing data quality. There may be a wide range of improvement actions involving different aspects of DIS design, varying from simple source selection (for example, if two sources provide the same type of data, the DIS can convey data from the source having the best quality) to changes in DIS design (for example, if a source is accessible only during certain periods, the DIS can materialize some data in order to enforce data availability).
- *Quality-driven DIS design*: This concerns the design of DISs taking into account quality guidelines, which may have the form of quality constraints or improvement actions.

Many quality factors have been proposed for modeling data quality, some of them defined in different manners. Several works propose classifications of quality factors according to semantic criteria [Wang+1996] [Naumann+2000], process-oriented criteria [Naumann+1999] [Weikum 1999] or goal-oriented criteria [Jarke+1997]. Other works propose formal frameworks for describing quality factors and deal with the management and storage of the appropriate metadata [Strong+1997] [Jarke+1997] [Jeusfeld+1998] [Helfert+2002] [Gertz 1998] [Missier+2001]. However, there is no consensus in the definition of quality factors. Each application domain has its specific vision of data quality as well as a battery of (generally ad hoc) solutions to solve quality problems [Berti-Equille 2004]. Furthermore, even if quality factors are frequently treated as being independent, there exist lots of relationships among them. The great number of quality factors and their inter-relationships cause quality evaluation to be a complex problem of many variables. Improving the quality of a system corresponds to optimizing a problem of N variables, which is of high complexity if done in a general context. As a consequence, it is difficult to consider many quality factors at a time. In order to study data quality in depth, we believe that it is necessary to study separately each quality factor as well as the properties of the environment that impact it. Afterwards, we can consider interaction between quality factors.

In [Wang+1996], the various quality attributes are analyzed from the user perspective. However, the definition of accurate profiles that allow users to understand the different quality metrics and to express their expectations is an open problem.

Regarding the assessment of source data quality, several works study and classify assessment techniques, types of metrics, units and aggregation functions [Pipino+2002] [Naumann+2000] [Ballou+1998]. For some quality factors, there are also detailed techniques for specific DISs scenarios, for example, the assessment of data accuracy in customers address attributes [Laboisse 2005]. Analogously, existing works treating the estimation of specific DIS properties (e.g. costs) can be reused for quality evaluation, for example, the estimation of data change frequency in caching systems [Cho+2003].

Despite the existence of several proposals for assessment of source data quality and assessment of DIS property values, putting such capabilities to use in DISs still requires modeling effort. There is a proposal for combining quality values of source data using simple arithmetic operations (minimum, maximum, average, sum and product) [Naumann+1999]. The need of an algebra for combining source quality values was also highlighted in [Gertz+2004]. Other works analyze the combination of source quality values for some specific quality factors, as accuracy and completeness, for example [Braumandl 2003] [Motro+1998]. But none of these works takes into account DIS property values.

The few proposals addressing quality-driven design deal with source selection, i.e. when several sources provide the same type of data, selecting the one whose data has the best quality or building a ranking of sources according to its quality [Mihaila+2000] [Naumann+1998] [Nie+2002] [Zhu+2002]. In [Naumann+1999], several query plans, accessing to different sources are compared, selecting the one having the best quality. Quality values are also taken into account in [Braumandl 2003] for selecting sources and servers for executing a user

query and monitoring the execution. An alternative approach consists in improving data quality by correcting data errors (e.g. typing errors) or reengineering DIS processes in order to avoid introducing errors (e.g. loose precision during calculations). Several data cleaning tools have been proposed, each one proposing a wide range of correction functions [Galhardas+2000] [Sattler+2000] [Raman+2001] [Vassiliadis+2001] [Lee+2000]. In [Ballou+1998], authors present some guidelines for DIS reengineering in order to improve the relation quality/cost of the information. Other works propose solutions for specific DIS scenarios, e.g. [Bright+2002] [Gancarski+2003]. But despite the existence of specific proposals, the land of quality improvement and quality-driven design is almost unexplored.

In this thesis we address the problems of evaluating the quality of the data delivered to users in response to their queries and deciding if users' quality expectations can be achieved. We also discuss how quality measures can be used for improving the DIS and enforcing data quality.

3. Our proposition

In order to introduce our proposal, we describe the technical issues addressed in this thesis and we present our contributions.

3.1. Technical issues addressed in this thesis

This thesis focuses on three main technical issues: (i) *analysis of quality factors and metrics*, (ii) *quality auditing,* and (iii) *quality improvement*.

Our approach consists in studying one quality factor at a time, analyzing its relationship with the DIS, proposing techniques for its evaluation and analyzing improvement actions for its enforcement. Among the quality factors that have been proposed, this thesis analyzes two main ones: *data freshness* and *data accuracy*. We summarize the analysis of each factor with a taxonomy, which allows comparing existent works and highlighting open problems. These taxonomies will serve to study the properties that impact the evaluation of each quality factor.

Concerning quality auditing, we propose the development of quality evaluation algorithms that take as input source data quality values and DIS property values generating as output a value for the data conveyed to the users. To this end, we model the different elements involved in data quality evaluation in a quality evaluation framework. Among these involved elements there are: data sources, user queries, DIS processes, DIS properties, quality measures and quality evaluation algorithms. In particular, we model DIS processes as workflow processes in which the workflow activities perform the different tasks that extract, integrate and deliver data to end-users. Quality evaluation algorithms are based on the workflow graph representation and consequently, the freshness evaluation problem turns into value aggregation and propagation through this graph. The idea behind the framework is to define a flexible environment, which allows specializing evaluation algorithms in order to take into account the characteristics of specific application scenarios.

Concerning quality improvement, we propose different kind of improvement actions to enforce data freshness and data accuracy when user expectations are not satisfied. Such actions are building blocks that can be composed to improve data quality in concrete DISs. For data freshness, we propose an enforcement approach that supports the analysis of the DIS at different abstraction levels in order to identify critical points (the portions of the DIS that are the bottlenecks for achieving freshness expectations) and to target the study of improvement actions for these critical points. The graph representation of the DIS allows the rapid visualization of critical points. For data accuracy, we propose the partitioning of query results in areas (some attributes of some tuples) having homogeneous accuracy. This allows user applications to retrieve only the most accurate data, to filter data not satisfying an accuracy threshold or to incrementally convey data (e.g. displaying first the most accurate areas). This represents an improvement to source selection proposals because accurate areas of several source relations can be combined while discarding inaccurate areas (instead of discarding whole sources).

Among the technical issues presented as motivation, we analyze freshness and accuracy factors, and we propose some solutions for quality auditing and quality enforcing problems. The proposed quality evaluation algorithms take as input the DIS processes and a set of values qualifying source data, DIS properties and user expectations. However, we do not deal with the acquisition of such values, which is carried out by quality assessment, property assessment and profile management techniques. Analogously, we propose basic improvement actions for enforcing freshness and accuracy in DISs, but we do not deal with the development of design (or reengineering) methodologies for adapting and combining the improvement actions in concrete DISs.

3.2. Main contributions

The main contributions of this thesis are:

- ***A detailed analysis of data freshness and data accuracy quality factors.*** The main result of this analysis is a survey on data freshness and data accuracy definitions and metrics used in the literature, which intends to clarify the meanings of such quality properties. We also elaborated a taxonomy of elements that influence quality evaluation. Additionally, this analysis highlights major research problems which still remain to be solved. As far as we know, such deep analysis has not yet been done for these quality factors.
- ***The proposal of techniques and algorithms for the evaluation and enforcement of data freshness and data accuracy.*** Our contribution consists in the specification of evaluation algorithms and improvement policies for data freshness and data accuracy. The definition of a homogeneous framework to manipulate quality factors constitutes a basis to the identification of the DIS properties that impact freshness and accuracy evaluation. It also allows an easy development of evaluation algorithms that consider such properties.
- ***A prototype of tool intended to be used in practical contexts of DIS management.*** The main results concerning the implementation of the proposed framework are the specification and prototyping of a quality evaluation tool that manages the framework. The framework components are specified in an abstract model, which supports the dynamic incorporation of new components to the tool, especially the inclusion of new quality factors and their evaluation algorithms. This brings support for the extensibility of the tool regarding the evaluation of other quality factors. The operational-style of the proposal, in particular the graph representation of DIS processes and the specification of quality evaluation algorithms as graph propagation methods facilitate their reuse for a wide range of DIS applications.

4. Outline of the thesis

The remaining of this thesis is organized in five chapters:

Chapter 2 presents the state of the art on the evaluation of data freshness and data accuracy quality factors. It provides a survey of freshness and accuracy definitions and their various underlying metrics proposed in the literature. We explore the dimensions that influence their evaluation, which are organized in taxonomies. Guided by the taxonomies, we analyze and classify the relevant work proposed for dealing with freshness and accuracy in different kinds of DISs. Both analysis, for data freshness and for data accuracy, show that existing work focus on specific types of DISs, specific characteristics (e.g. materialized data, some types of errors) and specific metrics, but other configurations remain untreated.

We identify several open research problems: specification of user expectations, acquisition of source data quality, formulation of cost models, data quality auditing and quality-driven engineering. We conclude by positioning our work with respect to these research problems; concretely, we focus on data quality auditing and quality-driven engineering issues.

Chapter 3 describes our proposal for data freshness evaluation and enforcement. We address two main problems: (i) evaluating the freshness of the data delivered to users in response to their queries, and (ii) enforcing data freshness when users' expectations cannot be achieved.

We propose a quality evaluation framework that models the different elements involved in data freshness evaluation, namely: data sources, user queries, DIS processes, DIS properties, quality measures and quality evaluation algorithms. In this framework, a DIS is modeled as a directed acyclic graph, called quality graph, which has the same workflow structure than the DIS and contains (as labels) the DIS properties that are relevant for quality evaluation. Quality evaluation is performed through evaluation algorithms that compute data quality traversing the quality graph and propagating property values. We discuss two kinds of propagations: (i) propagation of actual values, from sources to user interfaces, in order to calculate the freshness of delivered data, and (ii) propagation of expected values, from user interfaces to sources, in order to determine quality constraints for source providers. Therefore, we propose two propagation algorithms for data freshness. These algorithms take into account the DIS properties that impact data freshness, namely, the processing cost of DIS activities and the delays among them. The algorithms can be instantiated for different application scenarios by analyzing the properties that influence the processing costs and delays in those scenarios.

In addition, the framework proposes facilities to perform data freshness enforcement. If users' freshness expectations are not achieved, we may improve DIS design in order to enforce freshness or we may negotiate with source data providers or with users in order to relax constraints. The proposed enforcement approach allows analyzing the DIS at different abstraction levels in order to identify the portions that cause the non-achievement of freshness expectations. We suggest some elementary improvement actions, which can be used as building-blocks for specifying macro improvement actions adapted to specific DISs. As an application, we study the development of an improvement strategy for a concrete application scenario. The strategy consists in analyzing (and eventually changing) the execution frequency of DIS processes in order to satisfy user freshness expectations.

Chapter 4 describes our proposal for data accuracy evaluation and enforcement. We consider a relational context where user queries consist in selections, projections and joins over a set of source relations. We address two main problems: (i) evaluating the accuracy of the data delivered to users in response to their queries, and (ii) enforcing data accuracy when users' expectations cannot be achieved. Several algorithms and techniques that are used in the proposal are detailed in this chapter, as a complement to the state of art presented in Chapter 2.

Our approach for accuracy evaluation consists in two main phases: (i) partitioning source relations in areas with homogeneous accuracy in order to represent their distribution of inaccuracies, and (ii) for each user query, partitioning query result based on the partition of source relations. Each area is a virtual relation (view) over a source relation. User queries are reformulated (rewritten) in terms of areas. Specifically, the result of a user query consists in the union of a set of sub-queries, each one extracting data from several areas. We evaluate the accuracy of each sub-query and aggregate an accuracy value for the query result. We reuse the quality evaluation framework proposed for data freshness. We present an accuracy evaluation algorithm that takes into account the partitions of source relations and propagates them to query result. We focus on a priori evaluation, i.e. estimating data accuracy before executing user queries. Our algorithm explicitly indicates the areas that have lower accuracy, which allows filtering data not satisfying accuracy expectations.

Thereupon, we deal with accuracy improvement. We propose to use the partition of query result in order to select the areas that have the best accuracy. Note that we do not select whole relations but the portions that have the best accuracy, which differentiates our approach from the existing source selection approaches. We also proposed some basic improvement actions, for filtering data with low accuracy in order to satisfy several types of accuracy expectations.

Chapter 5 presents our experimentation results. Firstly, we describe a prototype of a data quality evaluation tool, DQE, which implements the framework. This tool allows displaying and editing the framework components as well as executing quality evaluation algorithms.

The prototype is used to evaluate data freshness and data accuracy in several application scenarios, enabling to validate the approach. Specifically, we describe three applications: (i) an adaptive system for aiding in the generation of mediation queries, (ii) a web warehousing application retrieving movie information, and (iii) a data warehousing system managing information about students of a university. We briefly describe each application, we model it in DQE (quality graphs, properties, etc.) and we explain the evaluation of data freshness and data accuracy for them.

Afterwards, we describe some tests for evaluating performance and limitations of the tool. To this end, we generated some data sets (quality graphs adorned with property values) and we executed a quality evaluation algorithm over each graph. The test results allow affirming that the tool can be used for large applications (modeling hundreds of graphs with hundreds of nodes each).

Chapter 6 presents conclusions and research perspectives.

Chapter 2. State of the Art

This chapter presents an overview on data quality evaluation in data integration systems and focus on the evaluation of two major quality factors: data freshness and data accuracy. We conclude with a discussion on open research problems.

1. Introduction

Traditionally, information quality is described or characterized via multiple attributes or factors which help to rank the data delivered to users (e.g. freshness, accuracy, completeness) or the processes that manipulate this data (e.g. response time, reliability, security). Many lists of quality factors have been proposed, some of them containing a great number of quality definitions. For example, Redman identified four dimensions of data quality: accuracy, completeness, currency and consistency [Redman 1996] while Wang and Strong analyzed the various quality attributes from the user perspective [Wang+1996].

However, there is no consensus in the definition of quality factors, which may be overlapping and contradicting. Each application domain has its specific vision of data quality as well as a battery of (generally ad hoc) solutions to solve quality problems [Berty-Equille 2004]. Furthermore, even if quality factors are frequently treated as being independents, there exist lots of relationships among them. Several works studied some relationships for specific systems, for example [Ballou+1995] [Bright+2002] [Cappiello+2002] [Theodoratos+1999] [Han+2003].

The great number of quality factors and their inter-relations cause quality evaluation to be a complex problem of many variables. To improve the quality of a system corresponds to optimize a problem of N variables, which may have a great complexity if done in a general context. As a consequence, it is difficult to consider all quality factors at a time. In order to study data quality in depth, it is necessary to study separately each quality factor as well as the properties of the environment that impact in it. Relationships should be studied thereafter.

Among the quality factors that have been proposed, we choose two of the main families: data freshness and data accuracy. The following sections describe both of them: Section 2 presents an analysis of data freshness and its various underlying metrics, describing some dimensions that impact freshness evaluation as well as the relevant works proposed for dealing with freshness in several kinds of DIS. Analogously, Section 3 analyzes data accuracy, also describing metrics, dimensions that impact its evaluation and relevant works.

The analysis of data freshness and data accuracy suggests open problems in the specification of user expectations, the acquisition of source quality measures and the formulation of cost models for the DIS. This knowledge can be used for developing techniques for data quality auditing and techniques for designing a system driven by quality expectations. An overview of such research problems is presented at the end of each section. We conclude, in Section 4, positioning our work with respect to those research problems.

2. Data freshness

Data freshness has been identified as one of the most important attributes of data quality for data consumers [Shin 2003] [Wang+1996]. Some surveys and empirical studies have proved that data freshness is linked to information system success [Wang+1996] [Mannino+2004] [Shin 2003]. Then, achieving required data freshness is a challenge for the development of a large variety of applications. Furthermore, the increasing need to access to information that is available in several data sources introduces the problem of choosing among alternative data providers and of combining data having different freshness values.

This section presents an analysis of data freshness and its various underlying metrics. We analyze some dimensions that impact the evaluation and enforcement of data freshness and we present a taxonomy that summarizes this discussion. The taxonomy is used for the analysis and classification of existing work. At the end of the section we discuss some open problems.

2.1. Freshness definitions

Intuitively, the concept of data freshness introduces the idea of how old is the data: Is it fresh enough with respect to the user expectations? Has a given data source the more recent data? Is the extracted data stale? When was data produced?

But data freshness has not a unique definition in the literature. Several freshness definitions have been proposed for different types of data integration systems. The traditional freshness definition, called *currency* [Segev+1990], is related to view consistency when materializing data and describes how *stale* is data with respect to the sources. Recent proposals incorporate another notion of freshness, called *timeliness* [Wang+1996], which describes how *old* is data. Therefore, freshness represents a family of quality factors, each one representing some freshness aspect and having its own metrics. For that reason freshness is commonly mentioned as a *quality dimension* [Jarke+1999]. Each factor best suites a particular problem or type of system. For example, in a replication system the number of refresh operations that have to be applied to a replica relation in order to reflect the changes made in a master relation is a good freshness metric, but in a data warehousing system the time passed from the update of an object to its delivery may be more relevant. In this sub-section we present an analysis of data freshness definitions.

We distinguish two quality factors for this quality dimension:

- *Currency factor* [Segev+1990]: The currency factor expresses how stale is data with respect to sources. Data is extracted from sources, processed (possibly stored for some time) and delivered to the users, but source data may have changed since data extraction and the user may receive stale data. The goal is to return the same data that is stored at sources so currency captures the gap between data extraction and data delivery. For example, currency indicates how stale is the account balance presented to the user with respect to the real balance at the bank database.
- *Timeliness factor* [Wang+1996]: The timeliness factor expresses how old is data (since its creation/update at the sources). The age of the data may be not appropriate for a given application. The goal is to return data that is valid (not too old) so timeliness captures the gap between data creation/update and data delivery no matter when data was extracted from sources. For example, when retrieving a top 10 CD ranking, timeliness indicates how old such ranking is: when was it created and stored in the CD seller source.

Figure 2.1 illustrates the gaps each factor aims to capture.

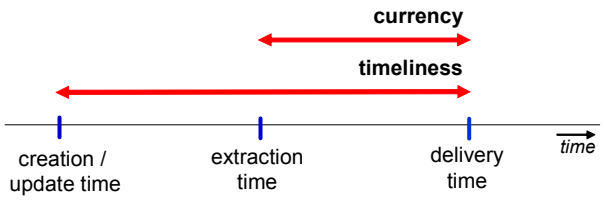

Figure 2.1 – Currency and timeliness factors

Note that both definitions are stated with respect to source databases. None of existing proposals compare with real world. The major argument is that capturing real world changes is not easy while capturing source changes is feasible (either querying, polling, subscribing…). However, for specific systems where real world changes are known both definitions can be enlarged to compare with real world data, i.e. how stale is delivered data (with respect to real world data) and how old is data (since its creation/update in real world). Examples of such systems are real-time registration of events (e.g. supermarket sales), sensor measurements and administrative documents (with specific timestamps).

2.2. Freshness measurement

A metric is a specific instrument that can be used to measure a given quality factor. There might be several metrics for the same quality factor. We describe the metrics proposed in the literature for measuring data freshness, classified by freshness factor:

- Metrics for the currency factor:
  - *Currency metric*: It measures the time elapsed since the source data changed without being reflected in the materialized view (if changes are propagated immediately then currency is 0) [Segev+1990]. In practice, as the precise change time may be difficult to obtain, currency is estimated as the difference between data extraction time and data delivery time. This estimation is used in data warehousing systems [Theodoratos+1999]. In caching systems it has been defined as *recency* or *age*, representing respectively the time elapsed since an object was cached [Bright+2002] and the time elapsed since an object became stale [Cho+2000]. In replication systems it is called *age* and measures the time since the oldest tuple of a relation has been waiting for the first refresh transaction [Gancarski+2003].
  - *Obsolescence metric*: It measures the number of updates to a source since the data extraction time. It can be measured from source logs or delta files or using change detection techniques [Hammer+1995]. Knowing the obsolescence of a source relation, the update frequency can be estimated and vice versa. Obsolescence is often called *age* in caching systems, meaning the number of times an object has been updated at the remote server since it was cached [Huang+1994]. In query processing systems, it is defined as the number of insertions, deletions and modifications since the data materialization time [Gal 1999]. In replication systems it is called *order* and measures the number of refresh transactions that have been committed in the master node but have not yet been propagated to the slave node [Gancarski+2003].
  - *Freshness-ratio metric*: It measures the percentage of extracted elements (tuples or attributes) that are up-to-date, i.e. their values equal the source relation ones. It can be estimated from the knowledge one has about data sources and from the way data is updated [Cho+2000]. It was defined as *freshness* in caching systems, meaning the number of elements of the cache that are up-to-date over the total number of elements [Cho+2000]. In [Labrinidis+2003], freshness is measured as the percentage of web pages that are fresh (not stale) in the cache but considering that the pages may have portions that are fresh and portions that are not.
- Metrics for the timeliness factor:
  - *Timeliness metric*: It measures the time elapsed since data was updated. It can be measured from source logs or delta files or using change detection techniques [Hammer+1995]. In [Braumandl 2003] it is measured from data timestamps, when they are available. Timeliness is generally estimated as the time elapsed from the last update to a source and bounded using the update frequency of the source data [Naumann+1999]. It was used in mediation systems [Naumann+1999] and web systems [Gertz+2004]. In [Ballou+1998], the metric is calculated as delivery time minus extraction time, adding the age of data at extraction time; some authors called it *currency*.

Previous data freshness metrics have been used as input for more complex indicators, for example, in [Ballou+1998] a freshness indicator is defined as[*]:

$$\max\{(1 - \text{timeliness} / \text{volatility}), 0\}^{s}$$

where volatility is measured in terms of shelf-life and the exponent *s* controls the sensibility of the ratio.

Table 2.1 presents a summary of freshness factors and their corresponding metrics.

Factor	Metric	Description
Currency	Currency	The time elapsed since data was extracted from the source (the difference between delivery time and extraction time).
	Obsolescence	The number of updates transactions / operations to a source since extraction time.
	Freshness ratio	The percentage of tuples in the view that are up-to-date (have not been updated since extraction time).
Timeliness	Timeliness	The time elapsed from the last update to a source (the difference between delivery time and last update time).

Table 2.1 – Summary of freshness factors and metrics

[*] In [Ballou+1998] the timeliness metric is called currency, and the composed indicator is called timeliness. We have changed their names for homogeneity purposes.

2.3. Dimensions for freshness analysis

In this sub-section we analyze some dimensions that impact the evaluation and enforcement of data freshness. We analyze the nature of data, the architectural techniques and synchronization policies of the underlying system. In next-subsection we present a taxonomy composed of these dimensions, which allows comparing different proposals for freshness evaluation.

Dimension 1: Nature of data

According to its change frequency, we can classify source data into three categories:

- *Stable data*: It is data that is improbable to change. Examples are scientific publications; although new publications can be added to the source, older publications remain unchanged. Other examples are person names, postal codes and country names.
- *Long-term-changing data*: It is data that has a very low change frequency. Examples are the addresses of employees, country currencies and hotel price lists in a tourist center.
- *Frequently-changing data*: It is data that has intensive change, such as real-time traffic information, temperature sensor measures and sales quantities.

The concept of "low frequency" is domain dependent; in an e-commerce application, if the stock of a product changes once a week it is considered to be low-frequency change while a cinema that changes its playbills weekly has a high-frequency change for spectators.

The nature of data is important because it is related to the notion of freshness that users are interested in. When working with frequently changing data, it is interesting to measure how long data can remain unchanged and minimize the delivery of expired data (i.e. evaluate currency). However, when working with stable or long-term changing data, these questions have no sense since data does not change very often. It is more interesting to measure how often new data is created or how old is the data (i.e. evaluate timeliness).

The changes can occur in a random way or with a defined frequency. For example restaurant menus, which are updated every morning, have a defined change frequency, but the account balances, which are updated with every account movement, have not got a defined frequency. In such cases, we can use data properties to develop specialized techniques, for example, synchronizing applications to extract data at the best moment.

Certain types of data have a lifecycle which describes explicitly its states and changing events. Some examples are the marital status of a person, the moon phases or the status of a semaphore. Sometimes, the events that make the states change are well known and can be predicted (as the semaphore). The fact that states are known in advance may allow the development of specialized techniques and treatments.

Dimension 2: Architectural techniques

The freshness of the data delivered to the user depends on *source data freshness* (the freshness of source data at extraction time) but also on the *execution delay* of the DIS processes (the amount of time from data extraction to data delivery). The DIS processes are very relevant in freshness evaluation because they can introduce significant delays. These delays may be relevant or not depending on freshness requirements. For example, in a given system, the evaluation of a query (minutes) is irrelevant compared to timeliness requirements (weeks), while in another system the aggregation processes may have the same order of magnitude of currency requirements (hours).

Specific cost models should take into account different parameters. These parameters depend on the system architectural techniques. We distinguish three main families of architectural techniques: those that calculate data when a new query is posed, those that cache the data most frequently used, and those that materialize the data needed to answer user queries. The features of these three categories of techniques are summarized below:

- *Virtual techniques*: The system does not materialize any data so all queries are calculated when they are posed. The system queries the relevant sources and merges their answers in a global answer that is delivered to the user. Examples are pure virtual mediation systems and query systems in database federations.
- *Caching techniques*: The system caches some information, typically data that is frequently accessed or the result of some frequent queries, and invalidates it when the time-to-live (TTL) has expired. If the

information required to answer a user query is stored in the cache, the system delivers it to the user; if not, the system queries the sources as in virtual systems. Examples are caching systems.

- *Materialization techniques*: The system materializes large volumes of data which is refreshed periodically. The users pose their queries and the system answers them using the materialized data. Examples are data warehousing systems and web portals that support materialization.

Virtual techniques allow to query sources and to return data immediately, so data is almost current. The processing and communication costs are the delays that influence currency. Caching techniques are conceived to return data as current as possible, estimating the TTL of each object for deciding when to invalidate it. However, materialized systems can tolerate some level of staleness. Data is stored for some time in the DIS repositories, which decreases its freshness; the refreshment frequency is an important delay.

Dimension 3: Synchronization policies

The way DIS are implemented influences the freshness of the data delivered to the users. Particularly, the synchronization among the sources, the DIS and the users has impact in data freshness because it introduces delays. For example, a DIS that synchronizes updates each end of the day may provide data which is not fresh enough with respect to the expectations of a given user.

According to the interaction among the DIS and the sources, the extraction processes can have *pull* or *push* policies. With pull policy, the DIS queries the sources to obtain data and with push policy, the source sends data to the DIS. In the latter, the notification of new available data can be sent by an active agent, for example initiated by a trigger, or can be determined by the DIS continuously polling the source. Active sources can have their own policies as sending each updated tuple, or sending sets of tuples every regular periods of time or when changes surpass a threshold. Pull policies can also be driven by temporal or non-temporal events.

According to the interaction among the DIS and the users, the query processes can also have pull or push policies. With pull policy, users directly pose queries to the DIS. With push policy users subscribe to certain queries and the DIS regularly conveys response data to the users. Pull and push policies can also be driven by temporal or non-temporal events.

Combining the previous interactions among users, DIS and data sources leads to six possible configurations which are shown in Figure 2.2. With synchronous policies, the user directly accesses source data. With asynchronous policies, the DIS answers user queries using materialized data and asynchronously, the materialized data is refreshed from source data. We name each configuration with the user-DIS policy followed by the DIS-source policy. Asynchronism is represented by a slash (/), synchronism by a dash (-):

- Pull-pull (arrow (a)): The interaction is fully synchronized. When a user poses a query (pull), it is decomposed and sent by the DIS to the sources (pull). This configuration is common in virtual mediation systems.
- Pull / pull (arrows (b) and (c)): When a user poses a query (pull) the DIS answers it using materialized data. Asynchronously, the DIS queries the sources in order to refresh materialized data (pull). It is common in data warehousing systems.
- Pull / push (arrows (b) and (e)): When a user poses a query (pull) the DIS answers it using materialized data. Asynchronously, the sources send data which refresh materializations (push). It is also used in data warehousing systems.
- Push / push (arrows (d) and (e)): When sources send data to the DIS (push), it is used to refresh the materializations. Asynchronously, the DIS conveys data to the users (push). It is used in publish/subscribe environments.
- Push / pull (arrows (d) and (c)): Materialized data is conveyed asynchronously to the users (push) and also asynchronously, the DIS queries the sources in order to refresh the materialized data (pull). It represents certain user applications (e.g. data marts) that are regularly fed from warehouse data.
- Push-push (arrow (f)): The interaction is synchronized. When sources send data to the DIS (push), the DIS conveys it to the users (push). This configuration is specific to some real time systems (alert systems) which capture events from sensors and conveys them to users but maintain also a history of these events. This configuration is not usually implemented in the three architectural techniques described previously.

Asynchronous policies introduce delays. The refresh frequency of the DIS repository is important to evaluate the freshness of retrieved data. When pushing data to the user, the push frequency is also important.

In systems where there are heterogeneous data sources with different access constraints and users with different freshness expectations, it is important to support and combine several kinds of policies.

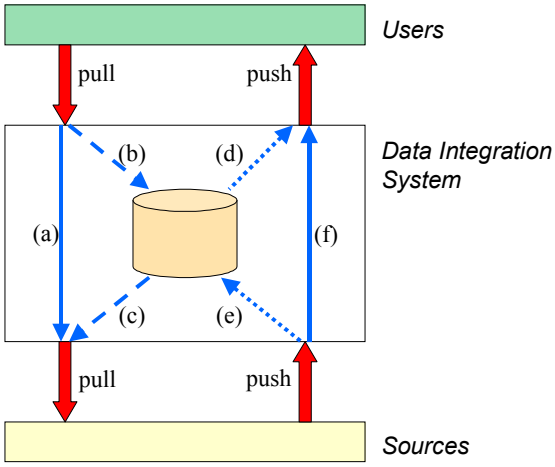

Figure 2.2 – Combination of synchronization policies

In next sub-section we present a taxonomy that relates these three dimensions summarizing their analysis.

2.4. A taxonomy for freshness measurement techniques

In this section we present a taxonomy that summarizes the discussion and allows comparing different proposals for freshness evaluation. The taxonomy is composed of the previously described dimensions: (i) nature of data, (ii) architectural techniques and (iii) synchronization policies.

Nature of data is a user-oriented dimension which qualifies the properties of source data from a user point of view. But not all the combinations of nature of data and freshness factors are interesting. On the one hand, when data changes very frequently, there is no interest in measuring timelines, which captures stable data behavior. On the other hand, there is no sense to evaluate the currency of long-term and stable data because they are almost always current as they do not change very often. In the latter case, the system can assure currency even without explicit evaluation. The other combinations need evaluation to determine the freshness level. For them, the development of evaluation tools is interesting. Table 2.2 shows the relation, indicating when data freshness can be assured without evaluation and when it is interesting to evaluate it (✓) or not (×).

Nature of data / *Freshness factor*	**Frequently changing**	**Long-term changing**	**Stable**
Timeliness	×	✓	✓
Currency	✓	assured	assured

Table 2.2 – Interesting combinations of freshness factors and nature of data

Architectural techniques, *synchronization policy* and *process complexity* are system-oriented dimensions which describe the system relation with data freshness. There is a correlation between *architectural techniques* and *synchronization policies*, since virtual techniques only support the synchronous pull-pull configuration, caching techniques are conceived for user pulls and materialization techniques support the asynchronous configurations. Table 2.3 shows the interrelations among them, indicating the valid combinations.

However, a priori, all combinations of *nature of data* and *architectural techniques* are possible, i.e. virtual, caching or materialization techniques (with their valid combinations of synchronization policies and process complexities) can be built to query different types of data.

Sync. policies / *Arch. techniques*	**pull-pull**	**pull/ pull**	**pull/ push**	**push/ pull**	**push/ push**
Materialized	×	✓	✓	✓	✓
Caching	✓	✓	✓	×	×
Virtual	✓	×	×	×	×

Table 2.3 – Valid combinations of architectural techniques and synchronization policies

In addition, system-oriented dimensions are also orthogonal to the freshness factors, i.e. users may be interested in both freshness factors, independently of the way the system is implemented. But taking into account the particularities of the systems and the availability of metadata and estimations, the different metrics are generally more related to some kinds of systems. For example, in virtual systems the main interest is the response time, so the currency metric is appropriate; in caching systems all the metrics have been identified as interesting, as different existent applications evaluate them [Bright+2002] [Huang+1994] [Cho+2000]; in materialized systems, currency and obsolescence have been used [Theodoratos+1999] [Gal 1999]. Table 2.4 shows the correlation among all the taxonomy dimensions and freshness metrics.

Nature of data / *Arch. techniques – Sync. policies*		**Frequently changing**	**Long-term changing**	**Stable**
Virtual	**Pull-pull**	Currency	Timeliness	Timeliness
Caching	**Pull-pull** **Pull/pull** **Pull/push**	Currency Obsolescence Freshness-ratio	Timeliness	Timeliness
Materialized	**Pull/pull** **Pull/push** **Push/pull** **Push/push**	Currency Obsolescence	Timeliness	Timeliness

Table 2.4 – Correlation of all the taxonomy dimensions

The technical problems to solve for each cell of the taxonomy are quite different. For example, enforcing currency in a materialized system implies developing efficient update propagation algorithms to deal with consistency problems, while evaluating timeliness in virtual systems is quite independent on the query rewriting algorithms and is dominated by source data timeliness. In Sub-section 2.6 we discuss evaluation problems.

2.5. Some systems that consider data freshness

In this sub-section we analyze several types of systems that evaluate freshness and we describe the goals and problems that they present. At the end of the sub-section, Table 2.5 summarizes the proposals in terms of the taxonomy presented before.

Data Warehousing systems

In data warehousing systems, freshness is studied through the *currency factor* in the context of view materialization.

The materialization of some views over source databases allows speeding up OLAP queries and reduces the overload of the sources. Traditional query optimization algorithms are extended to take into account the materialized views. In [Gal 1999], a cost model has been proposed for analyzing and comparing query plans, which can access to virtual and materialized data. The cost model strikes a balance between the query generation and data transmission cost on the one hand, and the *obsolescence* cost on the other hand.

Materialization introduces potential inconsistencies with the sources and warehouse data may become out-of-date [Hammer+1995]. The notion of consistency means that the DW state reflects an actual source state at some

"recent time" [Zhuge+1997]. The view maintenance problem consists in updating a materialized view in response to changes arisen at source data. Most of the work concentrates in assuring DW consistency for different types of views and refresh strategies. A classification of view maintenance algorithms is presented in [Gupta+1995]. A key problem in the last years has been the selection of a set of views to materialize in order to optimize the query evaluation and/or the maintenance cost, possibly in the presence of some constraints (commonly storage constraints). There are several works in this area [Harinarayan+1996] [Gupta 1997] [Theodoratos+1997] [Yang+1997] [Baralis+1997].

Data freshness is implicitly considered when defining the update propagation processes. Changes can be notified by active sources or can be monitored by the system accessing logs, querying the sources or comparing successive source snapshots [Widom 1995]. In most works, the update propagation processes are triggered by sources when the amount of changes is greater than a threshold or are executed periodically [Hull+1996] [Theodoratos+1997] [Baralis+1997] (pull/push and pull/pull policies). In [Theodoratos+1999], *data currency* is introduced as a quality factor in DW design. They propose an algorithm that takes as input the user expectations for data currency and determines the minimal update frequencies that allow achieving these values (pull/push policy).

Mediation systems

In classical mediation systems, freshness is also studied through the *currency factor*. In [Hull+1996], they formally define the concept of *guaranteed freshness*. A view is guaranteed fresh within a time vector, if it always corresponds to recent states of the source databases, that is, the difference between actual time and the time the view was calculated is always lower than the given time vector. Authors propose the construction of *Squirrel mediators*, which combine virtual and materialized data, and formally proof that they satisfy guaranteed freshness. Squirrel mediators combine pull-pull and pull/pull policies.

New proposals take into account the *timeliness factor*. It is used as a quality metric to compare among sources and to filter the data returned to the user. In [Naumann+1999], they introduce quality factors in a mediation system, which include *timeliness*. They study how to propagate a set of quality factors from several heterogeneous sources to the mediator. The propagation consists basically of merge functions that combine actual values of two sources through a relational operator. They propose a virtual scenario with pull-pull policy.

Caching systems

In caching systems, data is considered fresh when it is identical to data in the sources, so freshness is represented by the *currency factor*, and measured with the above mentioned metrics (currency, obsolescence, freshness-ratio).

An important problem is keeping cache data up-to-date. Traditional cache proposals manage the idea of *invalidation*. The system estimates the *time-to-live* (TTL) of an object as the time the object is supposed to be up to date, so the cache can store frequently changing data as well as long-term changing data. When the TTL has expired the object is invalidated in the cache, so the next access to the object will be directly read from the source and the cache will be refreshed (pull-pull and pull/pull policies). In some contexts the source can send invalidation information to the cache.

In [Cho+2000], they study the synchronization policies for cache refreshment and experimentally verify their behavior. They measure freshness with two metrics: currency (called age in the paper) and freshness-ratio. In [Li+2003a], they focus on the fine tuning of the caching policy of a dynamic content page cache, balancing response time and invalidation cycles for assuring data currency. In [Bright+2002], they propose the use of *latency-recency* profiles to adapt caching algorithms to user currency requirements, that is, if users demand more current data it will be extracted from the remote site paying communication times, but if currency of cached data is enough for user needs the user has an immediate response from the cache.

Newer proposals combine caching and materialization techniques. In [Labrinidis+2003], an adaptive algorithm has been proposed to combine view materialization and caching, balancing performance and data freshness. They combine a cache with server-side invalidation and a set of materialized WebViews (html fragments derived from a database) which are updated following a pull/push policy. As materialization degrades *currency*, the proposed algorithm selects which WebViews to materialize without exceeding a given currency threshold.

Replication systems

In a replication context, data at a slave node is totally fresh if it has the same value as the same data at the master node, i.e. all the refresh transactions for that data have been propagated to the slave node [Gancarski+2003]. Freshness is studied by means of the *currency factor* for frequently changing data.

A freshness model in a mono-master replication environment that supports OLTP transactions and OLAP queries has been presented in [Gancarski+2003]. Their goal is to determine the minimum set of refresh transactions needed to guarantee that a node is fresh enough with respect to the user freshness requirements for a given query. If data is not fresh enough, some refresh transactions are applied, then we can consider the mechanism as a cache invalidation method. The proposal consists in the evaluation of the freshness of slave nodes and the proposition of a load-balancing algorithm that takes freshness into account to decide when to refresh a replica. They follow pull-pull and pull/pull policies.

Works	Measurement	Nature of data	Arch. techniques	Synch. policy
Materialization for query processing [Gal 1999]	Obsolescence	Frequently changing	Virtual, materialized	Pull-pull, pull/pull
View maintenance [Gupta+1995] [Hammer+1995] [Zhuge+1997]	Currency	Frequently changing	Materialized	Pull/pull, pull/push
View maintenance policy [Theodoratos+1999]	Currency	Not specified	Materialized	Pull/pull
Selection of views to materialize [Harinarayan+1996] [Gupta 1997] [Theodoratos+1997] [Yang+1997] [Baralis+1997]	Currency	Frequently changing	Materialized	Pull/pull, pull/push
Mediation design combining virtual and materialized approaches [Hull+1996]	Currency	Not specified	Virtual, materialized	Pull-pull, pull/pull
Source selection in virtual mediation [Naumann+1999]	Timeliness	Not specified	Virtual	Pull-pull
Cache refreshment [Bright+2002] [Huang+1994] [Cho+2000] [Li+2003a]	Currency, obsolescence, freshness-ratio	Frequently changing / Long-term changing	Caching	Pull-pull, pull/pull, pull/push
Cache refreshment [Labrinidis+2003]	Freshness-ratio	Frequently changing	Caching, materialized	Pull-pull, pull/push
Replica refreshment [Gancarski+2003]	Currency, obsolescence	Frequently changing	Caching	Pull-pull, pull/pull

Table 2.5 – Summary of proposals

2.6. Research problems

Although data freshness has been studied in various ways in many papers, the analysis of the state of the art has shown that many problems remain unsolved or insufficiently treated. This sub-section summarizes these problems and mentions, when they exist, the references which have done significant contributions in each class of problem.

The freshness evaluation process needs to know about (i) the users' and source profiles, i.e. metadata about users' expectations and source properties, and (ii) the cost models used to extract source data, to maintain cached or materialized data and to evaluate query answers in different architectural configurations. Such elements should be adequately combined in order to evaluate the freshness of data conveyed to users. Freshness evaluation can be used for auditing an existing DIS or for designing a new DIS under quality constraints. In the following, we discuss these problems.

Defining users' and source profiles

Evaluating data freshness implies testing whether user's freshness expectations can be satisfied from source data freshness. One of the first problems is how and where to specify user expectations and how to define data source properties which impact data freshness.

Specification of freshness expectations

Users have freshness expectations for their applications which should be formulated according to some factors and metrics among those we have seen in Sub-sections 2.1 and 2.2. The specification of these factors and metrics pose a number of questions:

- Which language or formalism to use? Alternatives can vary from the simple specification of <property-value> pair [Bright+2002][Theodoratos+1999][Li+2003a] associated to each object type, to the definition of a specialized language if a preference order is introduced among freshness of different object types.
- At what level freshness expectations should be specified? We distinguish four levels: (i) for the whole system (for all users and data sources), (ii) for each data source (for all users), (iii) for each user or group of users, and (iv) for each user query. Each level implies different technical problems. When defining freshness expectations for the whole system or per source, individual user expectations should be reconciled. The expectations per user can be specified in a user quality profile. The definition of accurate profiles that allow users to understand the different freshness metrics and to express their expectations is an open problem. A first approach for introducing freshness in a user profile was presented in [Bright+2002]. Query languages such as *Preference SQL* [Kieβling+2002] can be extended to express freshness expectations in each user query.

Acquisition of source data freshness

The evaluation of data freshness at source level implies the selection of a freshness factor and the definition of metrics and measurement processes for it. The definition of new factors and metrics is an interesting area. Most existing works concentrate in the currency factor, but new types of DIS applications require the evaluation of other aspects of data freshness such as timeliness. Furthermore, several surveys have demonstrated the user interest in *timeliness* [Wang+1996] [Mannino+2004] [Shin 2003]. The acquisition of source data freshness poses some questions:

- Which source metadata is necessary to represent freshness in a source profile? In order to characterize source actual freshness some metadata should be obtained, for example the source update frequency [Cho+2003].
- How to acquire such metadata from sources? Some sources can provide useful information as the last update time or the update frequency, but for other sources these values must be learned or estimated from statistics elaborated during the data source exploitation. Existing techniques as comparing successive snapshots [Hammer+1995] or executing sampling queries [Jermaine 2003] can be adapted for freshness metadata acquisition. Techniques for specific metadata should be developed as in [Cho+2003].

Defining cost models

The freshness of the data delivered to the users depends on source actual freshness but also on the propagation delay from the sources to the user. Two main costs constitute this delay: the query evaluation cost and the update propagation cost. Depending on the taxonomy defined in Sub-section 2.4, these costs can be modeled differently:

Modeling of query evaluation cost: Query evaluation cost models for classical and distributed databases have been studied for a while and are well understood. Some proposals provide cost models for queries to cached or materialized data [Li+2003a] [Abiteboul+1998] [Cho+2003] and hybrid systems that combine materialization techniques in virtual or caching contexts [Labrinidis+2003] [Hull+1996]. Despite the existence of several proposals for specific systems, putting such capabilities to use in complex heterogeneous systems still requires modeling effort. Furthermore, existing cost models do not represent the cost of complex data processing involving ad hoc transformations and cleaning procedures such as in data warehousing systems. Several specialized tools, e.g. Ajax [Galhardas+2000] and Potter's Wheel [Raman+2001], require user interaction whose cost, although hard to estimate, should also be integrated.

Modeling of update propagation cost: Several cost models have been proposed to evaluate update propagation into materialized views [Chirkova+2001] [Yang+1997] [Hull+1996]. But as for query evaluation cost models, the update propagation cost models should be extended to represent complex workflow contexts with long transactions or interactive processes.

The challenge is the combination of all relevant parameters in a unified cost model in order to represent complex and hybrid architectures. Examples of such architectures are systems that extract data from heterogeneous sources with different synchronization policies and constraints. Some existing works combine several synchronization strategies [Segev+1990] [Theodoratos+1999] and temporal storage [Labrinidis+2003], but the land of hybrid systems is almost unexplored.

Several related questions are:

- Which DIS metadata is necessary to store in a DIS profile? Useful metadata will depend on the cost models used for modeling query evaluation cost and update propagation cost. For example, in [Hull+1996] several parameters are proposed for modeling the update propagation delay: announcement delay (amount of time between a source update and its announcement to the DIS), communication delay (amount of time needed for communicating with the source), update holding delay (amount of time between update announcement and start of update processing) and update processing delay (amount of time needed for processing an update). Large metadata should be studied for more complex cost models representing hybrid architectures.
- How to acquire such metadata from the DIS? For some metadata, as the query processing delay, acquisition techniques have been studied for classical and distributed databases [Abiteboul+1998] [Chirkova+2001]. Further metadata should be learned or estimated from statistics elaborated during DIS exploitation. Techniques for new types of metadata are also needed.

Auditing data freshness

Having users and source profiles and having cost models, one of the challenging problems is how to combine these elements in order to evaluate the freshness of the data conveyed to users. The main questions are: How should the different parameters of the source profile be combined for evaluating data freshness? What is the impact of update propagation cost and query cost in the freshness of data?

An additional question is how to combine several source actual values to obtain a global quality estimation. There are proposals only for specific environments. The combination of timeliness values within a virtual system is studied in [Naumann+1999]. The combination function is very simple (maximum of the input values) because they consider only simple views in a virtual context, but when dealing with materialization and complex calculation activities (which represent a bigger spectrum of DISs), other DIS properties (e.g. update propagation costs) should be considered. The combination of currency values within a materialized system is treated in [Theodoratos+1999]. They consider a DW architecture where source images are refreshed periodically and the propagation of changes to other materialized views follows push policies. Data currency is estimated adding the propagation cost of each materialized view and the refreshment period. But there are stills many combinations of freshness factors and types of systems for which there are no proposals. In particular, the combination of freshness values in hybrids systems, that manage data of different nature, different types of storage and synchronization policies is an open problem.

Freshness evaluation techniques can be used in the development of auditing tools, responsible for the evaluation of data quality. Several kinds of auditing tools can be conceived, for example:

- Prediction tools, for predicting the quality of data that can be returned in response to a query, without executing the query.
- Integrated evaluation tools, for measuring data quality during query execution and labeling delivered data with its quality levels. These tools can be integrated to the query evaluation process.
- Statistical tools, for taking samples of data quality during query execution and storing statistics. These tools can serve the first two categories.

The tools should use the query execution cost model and metadata describing the sources, the DIS and user expectations. They can be used at design time to evaluate the system, for example to test if user expectations can be achieved, or they can be used at run time for example, to predict the quality of the data delivered by alternative processes in order to choose the process that best suites user quality expectations.

Quality-driven engineering

Quality-driven engineering consists in designing a DIS under quality constraints. This kind of techniques may help in the selection among several design choices:

- Selection among alternative query plans that access alternative data sources.
- Specific optimization techniques as indexing and materialization
- Optimization of extraction and transformation algorithms.

Although several kinds of techniques have been explored in depth for specific systems, adapting their capabilities to heterogeneous systems requires addressing additional problems, as the combination of source data with different quality values, or even the comparison of data source profiles.

Obviously, quality-driven engineering techniques are based on quality auditing techniques, but the use of the latter ones may differ from evaluating an existing system and evaluating design alternatives. The challenge in quality-driven engineering is in the identification of the variables that influence data freshness and the proposition of techniques to achieve such improvements. There is few work done in this line, especially the study of synchronization policies [Cho+2000] [Li+2003a] [Segev+1990] or performance [Labrinidis+2003].

As improving freshness is not the unique quality goal for a given system, the relationship with other quality properties becomes a major challenge in DIS design. Research described in [Naumann+1999] has introduced quality factors to drive the design of virtual mediation systems; but quality factors are treated independently and only combined by means of a weighted sum.

The relationship among freshness and other quality properties has been only partially studied. There are two main lines: (i) tuning other properties in order to optimize freshness, and (ii) relaxing freshness in order to optimize other quality factors. In the former, the challenge is in the identification of related quality properties and the proposition of techniques that take advantages from these relationships in order to improve the global quality of data. Existing works in this line mainly concern the balance of freshness and performance [Labrinidis+2003] [Segev+1990] [Li+2003a]. In the latter, the expected freshness level is taken as a constraint. The main interest of existing works is performance or maintenance cost improvement [Bright+2002] [Gancarski+2003] [Theodoratos+1999] but the relation with other quality factors is still unexplored.

Discussion

Among the research problems previously mentioned, this thesis deals with data freshness auditing and freshness-driven engineering. Regarding data freshness auditing, we are interested in the combination of relevant parameters (source data freshness, cost models, user expectations) in order to evaluate the freshness of data conveyed to users. Users', sources and DIS profiles are taken as input. Contrarily to existing proposals [Naumann+1999] [Theodoratos+1999], we aim at considering the query evaluation and update propagation costs in a big spectrum of DISs, including those with data materialization and complex calculation activities. In addition, we are concerned with the tuning of DIS properties in order to enforce data freshness. Specifically, we focus in the proposal of improvement actions that can be used as building blocks for optimizing data freshness. Chapter 3 deals with data freshness evaluation and enforcement.

3. Data accuracy

Most data quality studies include accuracy as a key dimension for different types of DIS [Gertz+2004] [Shin 2003] [Vassiliadis+2000] [Mecella+2002] [Gertz+1998] [Missier+2001]. However, although the term has an intuitive appeal, there is no commonly accepted definition of what it means exactly. For example, in [Naumann+1999], accuracy is characterized as "the percentage of objects without data errors such as misspellings, out-of-range values, etc.", and in [Redman 1996], it is described as "the degree of agreement between a collection of data values and a source agreed to be correct". Then, accuracy represents a family of quality factors, or a quality dimension, with different associated metrics. Each factor best suites a particular problem or type of system.

This section presents an analysis of data accuracy and its various underlying metrics. We analyze some dimensions that impact the evaluation and enforcement of data accuracy and we present a taxonomy that summarizes this discussion. The taxonomy is used for the analysis and classification of existing work.

3.1. Accuracy definitions

Data accuracy is concerned with the correctness and precision with which real world data of interest to an application domain is represented in an information system [Gertz+2004]. Intuitively, the concept of data accuracy introduces the idea of how precise, valid and error-free is data: Is data in correspondence with real world? Is data error-free? Are data errors tolerable? Is data precise enough with respect to the user expectations? Is its level of detail adequate for the task on hand?

Data accuracy has not a unique definition in the literature. The most common definitions concern how correct, reliable and error-free is the data [Wang+1996] and how approximate is a value with respect to the real world [Redman 1996]. But there are various definitions (and variants of such definitions) concerning different concepts and metrics, which are mainly due to the different objectives of the systems where they are used. Furthermore, as data accuracy is highly related to other quality factors (e.g. data completeness, data consistency and data freshness), definitions are commonly confusing and overlapping. In this sub-section we present an analysis of data accuracy definitions.

Data accuracy comprises a family of quality factors, each one representing some accuracy aspect and having its own metrics. For that reason accuracy is commonly mentioned as a quality dimension [Jarke+1999]. We distinguish three quality factors for this quality dimension:

- *Semantic correctness factor*[*]: It concerns the correctness and validity degree of the data [Wang+1996] [Pipino+2002], describing how well data represent states of the real-world [Shanks+1999]. The concept of correctness implicitly or explicitly involves a comparison with the real world or with reference data agreed to be correct. It captures the gap (or the semantic distance) between data represented in the system and reference data. For example, the degree of concordance between the set of students that assist to a course and the list of students enrolled into this course.

  According to this definition, every set of data should represent a real world situation [Bobrowski+1998]. Different forms of incorrectness are: data without real world correspondent (mismembers), data referencing a wrong correspondent and data with erroneous attribute values [Kon+1995]. For example, data about a student may reference an inexistent person, reference an incorrect person or reference the correct person but having erroneous attributes (e.g. telephone).

- *Syntactic correctness factor*[†]: It expresses the degree to which a set of data is free of syntactic errors such as misspellings [Naumann+1999] and format discordances. Data is argued to be correct, in a syntactic way, if it satisfies syntactic rules and constraints imposed by users. Such constraints are not necessarily defined as integrity constraints of source databases, and generally cannot be added to data sources because of source autonomy; however, they correspond to the rules and constraints that DIS users expect that data verifies. Examples of constraints are: "inter-company telephone numbers have 4 digits" or "streets names must be registered in a street catalog". Typical constraints assure that data is always presented in the same format and is compatible with previous data [Wang+1996].

- *Precision factor*[‡]: It concerns the quantity of data to be stored and how precise this data must be [Redman 1996]. Data precision involves the level of detail of data representation [Bobrowski+1998] [Bouzeghoub+2002]. For example, the weight of a person may be stored in kilos (e.g. 88 Kg.) or with a greater precision (e.g. 88,25 Kg.). Analogously, a birthday may be represented by a year (e.g. 1973), by month and year (e.g. September 1973), a date (e.g. 19/9/1973) or even including time. In all cases there is a partial hierarchy inside each domain that allows deciding if a representation is more precise than another.

Note that considering the nature of the definitions, the semantic correctness and syntactic correctness factors attempt to quantify the quantity of data errors while the precision factor intends to quantify the precision of data. However, considering the type of comparison, the syntactic correctness and precision factors verify rule

[*] In the literature, this quality factor has been called *semantic correctness* [Missier+2001], *correctness* [Bobrowski+1998], *semantic accuracy* [Fugini+2002], *accuracy* [Wang+1996] [Gertz+2004] [Redman 1996] [Shanks+1999] [Wand+1996] [Naumann+2001] [Missier+2003] [Altareva 2004] [Gertz+1998], *free-of-error* [Pipino+2002], *soundness* [Motro+1998], *validity* [Müller+2003] and *data quality* [Ballou+1998]. Another alternative name is *precision* [Motro+1998].

[†] In the literature, this quality factor has been called *syntactic correctness* [Missier+2001], *correctness* [Missier+2003], *syntactic accuracy* [Fugini+2002], *accuracy* [Naumann+1999], *structural consistency* [Redman 1996], *representational consistency* [Wang+1996], *format consistency* [Missier+2003], *consistency* [Schurr+2002] and *schema conformance* [Müller+2003].

[‡] In the literature, this quality factor has been called *precision* [Bobrowski+1998] [Missier+2003] [Motro 1995], *accuracy* [Gertz+1998], *level of detail* [Redman 1996] and *ambiguity* [Wand+1996].

satisfaction focusing on syntactic aspects (data representation and constraints) while the semantic correctness factor compares to real world focusing on semantic aspects (data significance).

The following example will be used along the document:

Example 2.1. Consider an information system that handles information about students (Table 2.6) and consider real-world data about students (Table 2.7). Attributes describing students are: stid (the student identification number), name, address, telephone, interview (initial level determined by interviews; taking values 'high', 'medium' or 'low') and test (initial test result; taking values between 0 and 1). The key of the relation is {*stid*}.

Considering semantic correctness, stid 21 and 22 do not exist in real-world (mismembers), and stid 58 references the wrong student. Other students present matching errors or null values in some attributes (value inaccuracy), for example: student 43 has wrong address and telephone and student 102 has no registered address. However, note that even not written in the same way, some attribute values are good, e.g. the name of student 103 and the address of student 57. The address of student 56 is also ok; discrepancies are because the street name was changed some years ago, but postal office keeps the old name so address can be found.

Regarding syntactic correctness, note that some names and addresses do not follow standard rules, e.g. name of student 103 and addresses of students 21 and 57. The address of student 22 is not in an address catalog because of typing errors (*Coloniaa* instead of *Colonia*), and the test attribute of student 58 has an out of range value (9).

Finally, regarding precision, note that some test values are rounded (e.g. student 57) leading to lose of precision while other ones are more precise. The address of student 21 is also imprecise (*Carrasco* is the name of a neighborhood, not a detailed address). □

stid	name	address	telephone	interview	test
21	María Roca	Carrasco	6001104	low	1.0
22	Juan Pérez	Coloniaa 1280/403	9023365	medium	.5
43	Emilio Gutiérrez	Irigoitía 384	3364244	high	.8
56	Gabriel García	Propios 2145/101		low	.5
57	Laura Torres	Maldonado & Yaro	099628734	medium	.7
58	Raúl González	Rbla Rca Chile 1280/1102	4112533	high	9
101	Carlos Schnider	Copacabana 1210	094432528	high	.9701
102	Miriam Revoir		9001029	medium	.7945
103	A. Benedetti	Charrúa 1284/1	7091232	low	.9146
104	Luis López	Sixtina s/n		high	.8220

Table 2.6 – Students relation

stid	name	address	telephone	interview	Test
43	Emilio Gutiérrez	Potosi 934	6019945	high	.8000
56	Gabriel García	Batlle y Ordoñez 2145/101	5143029	low	.5130
57	Laura Torres	Maldonado 864	099628734	medium	.6965
58	Horacio Acher	Soca 2315	7079428	medium	.7600
59	Renzo Quinteros	Juan Paullier 635/001	4037690	low	.5505
101	Carlos Schnider	Copacabana 1210	094432528	high	.9701
102	Miriam Revoir	Canelones 1524	9001029	medium	.7945
103	Ana Benedetti	Charrúa 1284/1	7091232	low	.5146
104	Luis López	Sixtina s/n		high	.8218

Table 2.7 – Real world students

Other quality factors related to accuracy

Data accuracy is a quality measure for the relative amount of erroneous data accessed and manipulated by a system. In the past, accuracy was known as "data quality" and proposals devoted to evaluate and enforce data quality centered in accuracy (see for example [Ballou+1998]). For that reason, several accuracy definitions include other quality aspects as completeness or freshness, i.e. incomplete and expired data is considered inaccurate. In [Redman 1996], Redman distinguishes four quality dimensions for data values: *accuracy* (in the sense of correctness), *freshness* (in the sense of timeliness)[*], *completeness* and *consistency*, but remarks that accuracy is the most fundamental one, with freshness, completeness and consistency being special cases.

In this sub-section we discuss larger definitions of data accuracy and its relation with other quality factors.

Accuracy and Completeness

Completeness is commonly defined as the degree to which all data relevant to an application domain have been recorded in an information system [Gertz+2004]. It expresses that every fact of the real world is represented in the information system [Bobrowski+1998]. It is possible to consider two different aspects of completeness: (i) whether all required entities for an entity class are included, and (ii) whether all data values are present (not null) for required attributes. In [Redman 1996], the first aspect is called *entity completeness* and the latter *attribute completeness*, but other authors use different terms. Most works of literature define completeness as one or both of the previous definitions.

Some works consider accuracy and completeness as a same family of problems. In [Kon+1995], authors study error measurment and classify them into three categories: value innacuracy (errors or null values in some attributes), class mismembership (system data references inexistent real-world data) and class incompleteness (there is real-world data not referenced). They treat null values as a case of innacuracy and inexistent data as incompleness. Analogous definitions are presented in [Parssian+1999].

Accuracy and Consistency

Consistency expresses the degree to which a set of data satisfies a set of integrity constraints [Redman 1996]. Data is argued to be consistent if it satisfies the constraints[†]. Examples of constraints are: "the age of an employee accords its birthday year" (e.g. "Paul has 18 years" is inconsistent with "Paul was born in 1943") or "employee codes are unique". The most common constraints checks for null values, key uniqueness, duplicates and functional dependencies. In this sense, constraints state that two or more values do not conflict each other [Mecella+2002]. In particular, duplicate detection is one of the current challenges.

Semantic correctness (as previously defined) is very difficult to check, however, automated checking of constraints is feasible. In this direction, consistency checks provide a cost-effective way to identify data values that are not valid, knowing that satisfaction of constraints does not assure that data is semantically correct. Redman says that two data values are inconsistent if they cannot be correct simultaneously, so argues that defining consistency rules helps in the identification of data values that cannot be correct [Redman 1996]. The reader should note that such consistency rules may be defined for the integrated data (to be delivered to users), so, they may be not defined as constraints in the sources. While in traditional database systems all consistency rules are automatically checked by database engine and tuples not verifying them are rejected from the database, in DIS, consistency rules may represent expected rules (and may be not implemented as constraints), so the achievement of constraints can be measured as other quality factors (e.g. as a rate).

Accuracy and Freshness

As data values change with time, a lag between the time real-world data changes and the time changes are represented in the system is inherent. Data timeliness measures this lag, expressing the degree to which the recorded data are up-to-date.

[*] Timeliness is called currency in [Redman 1996]

[†] Analogously to syntactic constraints, integrity constraints are not necessarily defined in the source databases. They correspond to the constraints that DIS data is expected to verify.

Data changes have also an impact in data accuracy. As argued in [Redman 1996], a datum value is up-to-date if it is correct in spite of a possible discrepancy caused by time-related changes to the correct value; a datum is outdated at time *t* if it is incorrect at *t* but was correct at some time preceding *t*. In this sense, being out of date is simply a specific type of inaccuracy.

In next sub-section we analyze accuracy metrics for measuring accuracy quality factors.

3.2. Accuracy measurement

In this sub-section we describe accuracy metrics. We firstly introduce three types of metrics for measuring the accuracy of individual data items* and we discuss some aggregation functions for them; then, we explain the specific metrics that have been proposed in the literature.

We highlight three types of metrics:

- *Boolean metric*: It is a Boolean value (1=true, 0=false) that indicates if a data item is accurate (correct, precise) or not [Motro+1997].
- *Degree metric*: It is a degree that captures the impression or confidence of how accurate is data [Laboisse 2005]. Such degree is commonly represented in the [0-1] range.
- *Value-deviation metric*: It is a numeric value that captures the distance between a system data item and a reference one (e.g. its real-world correspondent entity) [Kon+1995]. Such distance is generally normalized to the [0-1] range.

In order to provide the user with a global accuracy measure for a whole relation (or a query result), an aggregated accuracy measure has to be synthesized. Given a set of data items S, with n elements (|S|=n)†, and a set of accuracy values $\{a_i\}$ where a_i is the accuracy value of the i-th element of S, $1 \le i \le n$, aggregation functions build a synthesized accuracy value for S. Typical aggregation functions are:

- *Ratio*: This technique calculates the percentage/ratio of accurate data of the system [Motro+1998]. This percentage is calculated as the number of accurate data items in the system divided by the total number of data items in the system. The accuracy of data items is expressed with Boolean metrics, i.e. $a_i \in \{0, 1\}$, $1 \le i \le n$. The accuracy of S is calculated as:

  $$Accuracy_{Ratio}(S) = |\{a_i / a_i = 1\}| / n$$

  A generalization can be done for the other types of metrics, considering the number of data items whose accuracy values are greater than a threshold θ, $0 \le \theta \le 1$:

  $$Accuracy_{Ratio}(S) = |\{a_i / a_i \ge \theta\}| / n$$

- *Average*: This technique calculates the average of the accuracy values of data items [Ballou+1998]. The accuracy of data items can be expressed with any type of metric. The accuracy of S is calculated as:

  $$Accuracy_{Avg}(S) = (\Sigma_i\ a_i) / n$$

  This technique is the most largely used, for the three types of metrics. Note that if accuracy of data items are Boolean values the aggregated accuracy value coincides with a ratio.

- *Average with sensibilities*: This technique uses sensibilities to give more or less importance to errors [Ballou+1998] and calculates the average of the sensitized values. Given a sensitivity factor α, $0 \le \alpha \le 1$, the accuracy of S is calculated as:

  $$Accuracy_{Sens}(S) = (\Sigma_i\ a_i^{\alpha}) / n$$

- *Weighted average*: This technique assigns weights to the data items, giving more or less importance to them [Ballou+1998]. Given a vector of weights W, where w_i corresponds to the i-th data item, $\Sigma_i\ w_i = 1$, the accuracy of S is calculated as:

  $$Accuracy_{weight}(S) = \Sigma_i\ w_i\ a_i$$

---

* Generally, the term *data item* refers to an attribute of a tuple (a cell), but in some application contexts it refers to the whole tuple.

† |A| denotes the cardinality (number of elements) of the set A.

Considering the three types of metrics and the aggregation functions, we describe the metrics proposed in the literature for measuring data accuracy, classified by accuracy factor.

- Metrics for the *semantic correctness* factor:
  - *Semantic correctness ratio metric*: It measures the percentage of semantically correct data in the system [Motro+1998]. This percentage is calculated as the number of system data items that match real world data divided by the number of system data items. This metric has been used in several works, for example [Shankaranarayan+2003] [Redman 1996] [Motro+1998].

    In practice, comparing all data against real world may be not viable, so correctness-ratio is commonly estimated via sampling. A portion of system data is taken as a sample, which is validated against real-world [Motro+1998] or a reference system considered reliable enough [Missier+2003]. For some types of data as postal addresses, telephone numbers or emails, there exist referential catalogs. When there are several available data sources, the most reliable one is usually taken as reference to perform the comparison; the appearance frequency could also be taken into account to decide whether data is correct or not [Shankaranarayan+2003] [Fugini+2002]. Other strategies consist in taking into account other quality factors to determine the most reliable data source, e.g. data consistency [Redman 1996]. Source providers or domain experts can also provide error ratio estimations based on their knowledge/experience with the data. Other specific methods are currently used to perform further validations [Laboisse 2005]; they include enquiries, automatic-generated emails or telephone calls to validate data with customers, even if such methods are very expensive and time consuming.

    Additional metrics have been defined to measure special cases of inaccuracies [Parssian+1999] [Kon+1995]:

    - *Mismembership ratio metric*: It measures the percentage of mismembers, i.e. the percentage of system data without correspondent in real-world.
    - *Value inaccuracy ratio metric*: It measures the percentage of system data containing errors in some attributes values or containing null values.
  - *Semantic correctness degree metric*: It captures the impression or confidence of how correct is data [Laboisse 2005]. The calculation can be done manually by a domain expert or it can be estimated based on historical data or statistics. Most common degrees are in [0-1] range (or translations) or any ad-hoc classification such as “good”, “medium” and “low”.

    This metric is typically used in automatic input-processing systems, e.g. optical character recognition, image recognition and address searching. For example, in optical character recognition applications, each symbol is compared to the alphabet characters; the most similar one is returned with its similarity degree. For example, for a certain application the three symbols of Figure 2.3 may be recognized as the character “C”, but the confidence of such recognition may be 80, 100 and 65 respectively.

    In order to compute the correctness degree of a set of elements, weighted average is typically used as an aggregation function, assigning different weights to the attributes according to their relative importance [Laboisse 2005].

Figure 2.3 – Character recognition examples

  - *Semantic correctness deviation metric*: It measures the semantic distance between a system datum and its real-world correspondent datum [Kon+1995] [Fugini+2002]. The calculation of such distance depends on the data type and the application, for example, for numeric data, it can be calculated as the difference between values (normalizing or not) [Shankaranarayan+2003] or for string data counting the number of characters to change (add, delete or modify) [Navarro 2001] or considering the soundness of words [USNARA 2000]. Values are generally normalized (translated to the [0-1] interval).

    In practice, if a comparison with real-world data is not possible, the comparison is done against a reference value which can be obtained from other data source or synthesized from several source values, e.g. using statistics of appearance frequency [Morey+1982] or taking an average value [Shankaranarayan+2003] [Fugini+2002].

For some attributes (belonging to a discrete domain), a similarity relationship among domain values may be defined, indicating to what extent each pair of labels resemble each other. When such relationship is defined, the similarity degree can be used as accuracy estimation. For example, Table 2.8 shows a possible similarity relationship for the interview attribute of the Students relation of Example 2.1, so the accuracy of the interview value of student 58 is 0.5. A similarity relationship can be defined by users according to its expectations or can be taken from some domain convention (e.g. color similarity ratios). Some types of fuzzy values (those of a discrete non-ordered dominion) have associated a similarity relationship (see [Galindo+2004] for a description of fuzzy values).

In order to compute the value-deviation of a set of elements, simple or weighted averages are typically used as aggregation functions, in the latter assigning different weights to the attributes according to their relative importance [Motro+1998].

	low	medium	high
low	1	0.5	0
medium	0.5	1	0.5
high	0	0.5	1

Table 2.8 – Similarity relationship among values of the interview domain

- Metrics for the *syntactic correctness* factor:
  - *Syntactic correctness ratio metric*: It measures the percentage of syntactically correct data of the system [Pipino+2002]. This percentage is calculated as the number of system data that satisfies syntactical rules divided by the number of system data. This metric is largely used in the literature; see for example [Naumann+2000] [Pipino+2002] [Naumann+1999].

    Note that when evaluating semantic correctness metrics, when the tuple is a mismember, all the attribute values are inaccurate and when the key is not accurate most of the attribute values are inaccurate (except for hazard coincidences). This is the desired effect. So, semantic correctness metrics evaluate if an attribute corresponds to the real world object represented by the key. However, when evaluating syntactic correctness, the metric semantics depends on the precise rules; for example, syntax constraints or belonging to a range can be checked independently of the key attributes.

    The most typical syntactical rules checks for illegal values (e.g. out-of-range), non-standard format or embedded values (e.g. “Paris, France” in a *city* attribute).

  - *Syntactic correctness deviation metric*: It measures the syntactic distance between a system datum and some neighbor data that is syntactically correct [Fugini+2002]. The calculation of such distance (and neighbor determination) depends on the type of constraint, the data type and the application, for example, for string domain conformity checks (data belongs to a catalog) the distance to most similar element can be used [Navarro 2001] [Gravano+2001] [USNARA 2000] or for out of range checks the distance to the nearest range can be calculated. As an example, consider the *Name* attribute of Table 2.7 as a reference catalog for students’ names. The nearest element for “A. Benedetti” is “Ana Benedetti” and the value-deviation metric can be calculated using some edit distance function.

    In order to compute the value-deviation of a set of elements, simple or weighted averages are typically used aggregation functions, assigning different weights to the attributes according to their relative importance [Laboisse 2005].

- Metrics for the precision factor:
  - *Scale metric*: For numeric values, precision is commonly associated to the measurement scale (or error-ratio of the measurement instrument). For example, if the length of a certain table is 87 ±1 cm, then imprecision is ±1 cm. A relative imprecision metric can be obtained dividing by the data value, i.e. 1/87; the precision metric is obtaining subtracting imprecision from 1, i.e. 1 – 1/87. When the error-ratio is not given, a confidence degree in the measurement process can be used, generally measured as the units of the least significant digit of a measurement, e.g. in 17,130 meters imprecision is millimeters (0,001 m) [Wikipedia 2006].
  - *Standard error metric*: In contexts where a data item is obtained taking several measures of some phenomena (e.g. temperature or traffic flow), imprecision is usually characterized in terms of the standard deviation of the measurements, called the measurement process's standard error [Wikipedia

2006]. This metric, even largely used in scientific environments, is rarely used in data integration systems because metadata about data production is not frequently available.

- *Granularity metric*: It involves the number and coverage of attributes that are used to represent a single concept [Redman 1996]. For example, an address may be represented by only a country name (e.g. Uruguay) or by a set of attributes like street name, door number, city, postal code and country (e.g. Cubo del Norte, 3840, Montevideo, 11700, Uruguay). The second set of attributes provides a more granular view of data. A simple metric consists in counting the data values (non-null values) representing a concept.

Table 2.9 summarizes accuracy factors and their corresponding metrics.

Factor	Metric	Description
Semantic Correctness	Semantic correctness ratio	The percentage of system data that match real-world data.
	Mismembership ratio	The percentage of system data without correspondent in real-world.
	Value inaccuracy ratio	The percentage of system data containing errors in some attributes representation.
	Semantic correctness degree	The confidence (degree) on the correctness of data
	Semantic correctness deviation	The semantic distance between a system datum and its correspondent real-world datum.
Syntactic Correctness	Syntactic correctness ratio	The percentage of system data that satisfies syntactical rules.
	Syntactic correctness deviation	The syntactic distance between a system datum and a reference one considered as syntactically correct.
Precision	Scale	The precision associated to the measurement scale.
	Standard error	The standard deviation of a set of measurements.
	Granularity	The number of attributes used to represent a single concept.

Table 2.9 – Summary of accuracy factors and metrics

3.3. Dimensions for accuracy analysis

In this sub-section we analyze some dimensions that impact the evaluation and enforcement of data accuracy. We analyze the granularity of the measurement, the typology of errors, the data types and the architectural techniques of the underlying system. In next-subsection we present a taxonomy composed of these dimensions, which allows comparing different proposals for accuracy evaluation.

Dimension 1: Granularity of measurement

The techniques for accuracy acquisition may vary according to the evaluation granularity. Traditionally, a global accuracy measure was used to qualify a whole relation [Kon+1995] or a view over a relation [Naumann+1999]. In addition, some measurement techniques consider the relationship among relations [Rahm+2000]. Recent proposals, especially industrial works, focus on the measurement of the accuracy of individual cells* [Laboisse 2005]. We describe these three levels of granularity:

- *Cell granularity*: In this approach, an accuracy value is associated to each cell, measured either as a Boolean value (accurate / not accurate), a deviation from the correct value or a degree of accuracy. The accuracy measure is generally obtained evaluating each cell of the relation. This is a very costly task but industrial investment in accuracy measurement and improvement [Laboisse 2005] [Amat+2005] [Graveleau 2005] indicate its importance despite the cost.

* The term *cell* refers to an attribute of a tuple.

- *Set granularity*: In this approach, a unique accuracy value is estimated for a whole relation or view, typically measured as the number of accurate cells over the number of cells (ratio metric). The accuracy measure can be obtained evaluating each cell of the relation or each cell of a sample; statistical techniques can also be used. In several cases, the experience or confidence of experts can be used as estimation.
- *Relationship granularity*: In this approach, a unique accuracy value is associated to the relationship among relations or views, typically measured as the number of accurate cells over the number of cells (ratio metric). The accuracy measure is generally obtained applying arithmetic operations to the accuracy of relations or views (e.g. multiplying such values [Naumann+1999]).

The main problem of associating an accuracy value to a whole relation is that it does not show where inaccuracies are concentrated (some attributes, some sets of tuples). As argued in [Motro+1998] information sources are rarely of uniform quality, so a unique accuracy value may be a very crude estimation of the accuracy of specific data. On the other hand, cell granularity is sometimes too detailed, as sets of attributes or tuples may have the same behavior. This causes great overhead in measurement and excessive additional storage.

An intermediate solution consists in partitioning the source relation in areas that are highly homogeneous with respect to their accuracy [Motro+1998] and assigning an accuracy value to each area (thus remaining in set granularity). Homogeneity means that any sub-area of a highly homogeneous area would maintain roughly the same accuracy as the initial area. Areas are defined as views, which may involve selection and projection. Partitioning may be done by source providers considering their knowledge about data, for example which attributes are more probable to have errors or which sets of tuples (e.g. those corresponding to foreign sales) are more probable to be inaccurate. In [Motro+1998], Motro and Rakov propose an algorithm to perform partitioning in an automatic way, testing different partitioning criteria. Even if the algorithm has a great complexity, heuristics can be considered for specific cases. Furthermore, authors argue that the partitioning could be done only once, at design time, so the cost may be tolerated. The accuracy measure for each area can be obtained in the same way as for the whole relation.

Example 2.2. Consider the *Students* relation of Example 2.1 and the measurement of semantic correctness on it. The source database administrator provides the following information:

- Addresses and telephones of old students (id < 100) is very inaccurate (accuracy=0.25) because of data obsolescence.
- Stid, name, interview and test of old students have accuracy of 0.50 due to typing errors and lack of automated checking in an old information system.
- Data about new students (id ≥ 100) is almost accurate (accuracy=0.90).

A horizontal partition with two areas can be defined, with the following predicates: (i) stid < 100, and (ii) stid ≥ 100. The first area is also vertically partitioned in two sub-areas for the {stid, name, interview, test} and {address, telephone} sets of attributes respectively. Table 2.10 illustrates the partitioning coloring areas (and sub-areas) with different colors. □

stid	Name	address	telephone	interview	test
21	María Roca	Carrasco	6001104	low	1.0
22	Juan Pérez	Coloniaa 1280/403	9023365	medium	.5
43	Emilio Gutiérrez	Irigoitía 384	3364244	high	.8
56	Gabriel García	Propios 2145/101		low	.5
57	Laura Torres	Maldonado & Yaro	099628734	medium	.7
58	Raúl González	Rbla Rca Chile 1280/1102	4112533	high	.9
101	Carlos Schnider	Copacabana 1210	094432528	high	.9701
102	Miriam Revoir		9001029	medium	.7945
103	A. Benedetti	Charrúa 1284/1	7091232	low	.9146
104	Luis López	Sixtina s/n		high	.8220

Table 2.10 – Partition of the Students relation ($\pi_{address, telephone}(\sigma_{stid < 100}(Students))$, $\pi_{stid, name, interview, test}(\sigma_{stid < 100}(Students))$, and $\sigma_{stid \geq 100}(Students)$)

Regarding storage, cell granularity needs more storage space. Accuracy values are stored for each cell, for example in an accuracy matrix [Motro+1998] where columns represent attributes, rows represent tuples and values in the table correspond to the accuracy of the tuple attribute (cell). In the proposal of [Motro+1998], authors store Boolean metrics of semantic correctness but the matrix can be used with the other types of metrics and factors.

Example 2.3. Continuing Example 2.1, Table 2.11 and Table 2.12 show two accuracy matrixes for the *Students* relation, both measuring semantic correctness, the former with a Boolean metric and the latter with a value-deviation metric. The value deviation metric is calculated as the distance between the actual value (v) and the database one (v'), normalized and subtracted from 1. For the test attribute (numeric value), the arithmetic difference is used as distance and the actual value is used for normalization:

$$\text{accuracy}(v',v) = \max\{(1 - |v - v'| / v), 0\}$$

For the name attribute (string value), the string edit distance is used, counting the number of characters that must be changed (added, deleted or modified) in order to transform the database value into the actual one; the size of the actual value is used for normalization:

$$\text{accuracy}(v',v) = \max\{(1 - \text{string\_distance}(v - v') / \text{size}(v)), 0\}$$

For the address attribute, a GIS application is used in order to determine if addresses match. A value of 1 corresponds to addresses that completely match (no matter typing errors), 0 corresponds to addresses that do not match and intermediate values correspond to addresses that partially match (e.g. same street but different door-numbers). Note that incomplete addresses can be useful (for example, letters can arrive to destination), however little errors in telephone numbers causes that the person cannot be contacted; so, for the telephone attribute, the distance is calculated by a Boolean function. For the interview attribute (enumeration with values low, medium and high), their similarity degrees are used as distance (given in Table 2.8). □

	stid	name	address	telephone	interview	test
21	0	0	0	0	0	0
22	0	0	0	0	0	0
43	1	1	0	0	1	1
56	1	1	1	0	1	1
57	1	1	1	1	1	1
58	1	0	0	0	0	0
101	1	1	1	1	1	1
102	1	1	0	1	1	1
103	1	1	1	1	1	0
104	1	1	1	1	1	1

Table 2.11 – Accuracy matrix of the Students relation (semantic correctness, Boolean metric)

	stid	name	address	telephone	interview	test
21	0	0	0	0	0	0
22	0	0	0	0	0	0
43	1	1	0	0	1	1
56	1	1	1	0	1	0.9974
57	1	1	0.5	1	1	0.9950
58	1	0	0	0	0.5	0.8444
101	1	1	1	1	1	1
102	1	1	0	1	1	1
103	1	0.8333	1	1	1	0.5627
104	1	1	1	1	1	0.9998

Table 2.12 – Accuracy matrix of the Students relation (semantic correctness, value-deviation metric)

An alternative storage way is extending the relation with additional attributes, each attribute for storing the accuracy of each data attribute. For example, Table 2.13 shows the *Students* relation with additional attributes (A_{sti}, A_{nam}, A_{add}, A_{tel}, A_{int} and A_{tes}) which store the accuracy values corresponding to the stid, name, address, telephone, interview and test attributes respectively.

For the set granularity only an accuracy value is stored for the whole relation or view. Following previous example, a semantic correctness ratio of 0.6 (ratio metric) or an average of 0.6247 (value-deviation metric) can be stored for the Students relation. In the particular case of partitions (areas are views), an accuracy value is stored for each area. For example, the semantic correctness ratios of 0.25, 0.50 and 0.9 are associated to the areas defined in Example 2.2 respectively. For the relationship granularity an accuracy value is stored for the relationship.

stid	A_{sti}	name	A_{nam}	address	A_{add}	telephone	A_{tel}	interview	A_{int}	test	A_{tes}
21	0	María Roca	0	Carrasco	0	6001104	0	low	0	1.0	0
22	0	Juan Pérez	0	Coloniaa 1280/403	0	9023365	0	medium	0	.5	0
43	1	Emilio Gutiérrez	1	Irigoitía 384	0	3364244	0	high	1	.8	1
56	1	Gabriel García	1	Propios 2145/101	1		0	low	1	.5	1
57	1	Laura Torres	1	Maldonado & Yaro	1	099628734	1	medium	1	.7	1
58	1	Raúl González	0	Rbla Rca Chile 1280/1102	0	4112533	0	high	0	.9	0
101	1	Carlos Schnider	1	Copacabana 1210	1	094432528	1	high	1	.9701	1
102	1	Miriam Revoir	1		0	9001029	1	medium	1	.7945	1
103	1	A. Benedetti	1	Charrúa 1284/1	1	7091232	1	low	1	.9146	0
104	1	Luis López	1	Sixtina s/n	1		1	high	1	.8220	1

Table 2.13 –Students relation enlarged with accuracy values (semantic correctness, Boolean metric)

Dimension 2: Typology of errors

As measuring accuracy corresponds to quantifying data errors or data imprecisions, measurement techniques are closely related to the types of errors or anomalies that arise to data. We use the term *error* in a wide sense, including not only incorrect and malformed values but also values that do not follow user expectations, for example, values that do not follow certain representation standard.

Several error classifications have been proposed in the literature, particularly in the domains of data cleaning and data mining [Rahm+2000] [Oliveira+2004] [Müller+2003] [Quass 1999] [Berti-Equille 2004]. We grouped such types of errors in 7 categories:

- *Value errors*: This category encloses syntactical errors in cells, such as out-of-range values, misspellings and other typing errors.
- *Standardization errors*: This category includes values that are syntactically correct but do not follow an expected standard, e.g. standard format or standard units.
- *Embedded values*: This category represents cell values that correspond to multiple values (e.g. an address attribute embedding street, door number, city and postal code).
- *Missing values*: This category corresponds to dummy or null values for mandatory attributes.
- *Integrity rule violations*: This category encloses violations to integrity rules among attributes and among tuples, such as functional dependencies or uniqueness constraints.
- *Duplicates*: This category corresponds to values that are duplicated in the database, either consistently or contradictorily.
- *Wrong values*: This category represents values that do not correspond to real-world entities.

Table 2.14 presents a summary of error types proposed in the literature, detailing a name for the problem*, a brief description and an example; the type column corresponds to our classification.

The classification of missing values is debatable; they can also be considered as value errors or integrity rule violations. We have chosen to classify them in a separate category because many works concentrate in null values, for example [Naumann+1999] [Redman 1996]. However, note that null values are generally related to the consistency and completeness quality factors but not to accuracy; to the former because having null values for mandatory attributes violates data integrity and to the latter because they cause incomplete descriptions of real world entities. Similarly, *integrity rule violations* and *duplicates* categories do not refer to accuracy but consistency problems, i.e. errors in the relationships among values and violations of dependencies among them. We described them here in order to be consistent with existent error classifications but we will omit them in the rest of the section.

Type	Problem	Descripcion	Example
Value errors	Illegal values	Values outside of domain range	date = 30/13/70
	Misspellings	Values incorrectly written; usually typos and phonetic errors	city=”Paaris”
	Misfielded values	Value entered in the wrong attribute	city=”France”
Standardization errors	Cryptic values	Cryptic values and abbreviations	occupation=”DB Prog.”
	Use of synonyms	Expression syntactically different but semanti-cally equivalent	occupation=”professor” occupation=”teacher”
	Word transpositions	Word transpositions, usually in a free-form field	name= “J. Smith” name=”Smith J.”
	No standard format	Values appear in different formats	date=”13/08/1974” date=”1974/08/13”
	No standard units	Values appear in different units	value=123 (euros / dollars)
Embed-ded values	Embedded values	Multiple values entered in one attribute (e.g. in a free-form field)	name=”J. Smith 12.02.70 New York”
Missing values	Missing values	Dummy or null values for mandatory attributes	phone=9999-999999
Integrity rule violations	Attribute dependency violations	Attribute dependency violation	age=22; date=12/02/70
	Uniqueness violations	Uniqueness violation	emp1=<”John Smith”,832> emp2=<”Peter Miller”,832>
	Referential integrity violations	Referential integrity violation	emp1=<”John Smith”, D127> department D127 not defined
Duplicates	Duplicated records	Same information represented twice (e.g. due to some data entry errors)	emp1=”John Smith” emp2=”J. Smith”
	Contradicting records	Same information is described by different (contradicting) values	emp1=<”John Smith”, 12/02/70> emp2=<”John Smith”, 22/02/70>
Wrong values	Incorrect values	Value does not correspond to real world situation	age=35 actual age is 37
	Wrong references	Referenced value is defined but wrong	emp=<”John Smith”, D127> actual department is D157

Table 2.14 – Typology of errors

Dimension 3: Data types

Measurement techniques are closely related to data types. Example 2.3 motivated this fact, where specific formulas were proposed in order to calculate value deviations for different data types. In the literature, most efforts are dedicated to evaluating distances among numeric values [Ballou+1998] and among string values

* When different names were proposed we take the most frequently used.

[Navarro 2001] [USNARA 2000], but some works propose specialized techniques for particular cases, such as addresses, telephones and emails [Amat+2005].

We observe data types from the user point of view, i.e. which is the type of the data that users expect. Note that data types may be different at sources, e.g. a date may be embedded in a string attribute. Furthermore, different sources may provide the same data but with different data types; for example, sex may be represented by integer numbers (0 and 1) and by strings ("male", "female", "M", "F", etc.). As a consequence, it may be possible that source data does not comply with the data types expected by users. In some cases appropriate transformation functions can be applied in order to standardize data (e.g. sex attribute of previous example), but in other cases, as in free-form strings, functions are very hard to abstract (e.g. free form address attributes). The most detailed the data types are specified, the easier data errors and anomalies may be identified and corrected.

We propose a classification of data types in some basic categories and discuss some special cases for each one:

- *Numeric types*: They constitute the easiest data type to check for imprecisions and deviations, allowing the definition of precise arithmetic functions to estimate data accuracy. As a special case, range specification allows detecting further anomalies.
- *Date types*: As dates are precisely typed, it is also easy to check for invalid values (e.g. 30/02/2006), format discrepancies (e.g. 02/20/2006 when expecting DD/MM/YYYY format) and imprecise values (e.g. February-2006 when also expecting the day). Deviations can be easily calculated with standard date-difference functions.
- *Enumeration types*: They enclose all data types representing a finite set of elements, e.g. month names, which can be mapped to naturals. Out of range checking is easy; the distance to the closer element can be used to correct errors (e.g. transform "Fevruary" into "February"). Lots of special cases can be defined, generally using term glossaries, domain ontologies or reference catalogs for comparing with valid elements. Most common examples are code lists (e.g. airport international codes) and name catalogues (e.g. street names). The increasing definition of domain ontologies allows checking for belonging of additional attributes, as technical terms (e.g. disease names) and the detection of synonyms and abbreviations. Mapping functions may also be defined in order to transform from an enumeration type (used at a source) to another one (expected by users), for example for translating technical terms from English to Spanish.
- *String types*: They constitute the most widely used data type and the most difficult to check for errors. In some special cases, attribute domains may be described by grammars, which allow the detection of value errors (e.g. personal names following the format "name initial dot surname" as "J. Smith"). Free form strings are very hard to treat and generalize; embedded values and lists of elements may appear. However, many special cases have been sufficiently treated for some specific domains, for example, customer addresses, emails and telephones in CRM* applications.

Further data types can be defined for specific applications, for example, streams, pictures, video, etc.

Dimension 4: Architectural Techniques

The accuracy of the data delivered to the user depends on *source data accuracy* (the accuracy of source data at extraction time) but also on the errors introduced or corrected by DIS processes. The *architectural techniques* dimension discussed for data freshness also have impact in data accuracy because some detection and correction techniques can be executed only in particular environments. We recall the three main families of architectural techniques discussed in Sub-section 2.3:

- *Virtual techniques*: The system does not materialize any data so all queries are calculated when they are posed. The system queries the relevant sources and merges their answers in a global answer that is delivered to the user. Examples are pure virtual mediation systems and query systems in database federations.
- *Caching techniques*: The system caches some information, typically data that is frequently accessed or the result of some frequent queries, and invalidates it when the time-to-live (TTL) has expired. If the information required to answer a user query is stored in the cache, the system delivers it to the user; if not, the system queries the sources as in virtual systems. Examples are caching systems.

* CRM = Customer Relationship Management

- *Materialization techniques*: The system materializes large volumes of data which is refreshed periodically. The users pose their queries and the system answers them using the materialized data. Examples are data warehousing systems and web portals that support materialization.

When materializing data, complex transformations can be applied to data, including routines for error correction or format standardization. Such processes can also introduce errors, for example, rounding numbers can lose precision or normalizing addresses can cause values to be not longer semantically correct. Virtual techniques must provide query answers within short delays, so such complex transformation cannot be applied, however, simple format transformations and arithmetic operations are sometimes included in such systems. So even in less degree, some errors can be introduced by virtual techniques. Caching techniques only copy data from sources, so no errors are introduced no corrected during the process.

In addition, some measurement techniques, especially those comparing to real-world, which needs greats amounts of time for executing, only can be implemented for materialized data.

In next sub-section we present a taxonomy that relates these four dimensions summarizing their analysis.

3.4. A taxonomy of accuracy measurement techniques

In this sub-section we present a taxonomy that summarizes the discussion and allows comparing different proposals for accuracy evaluation. The taxonomy is composed of the previously described dimensions: (i) granularity of measurement, (ii) typology of errors, (iii) data types, and (iv) architectural techniques.

Firstly, there is a close relation between the *typology of errors* and the accuracy factors, since factors allow quantifying data errors, i.e. inaccuracies. Some relations are quite intuitive; semantic correctness is related to wrong values (values that do not match real world situations) and syntactic correctness is related to value errors, standardization errors and embedded values (all of them representing syntactic errors). However, the relation of the precision factor with some types of errors is less evident to see. Commonly, lack of precision appears as a standardization error (e.g. 143 when expecting a value with two decimal digits). In addition, some value errors, specifically out of domain values (e.g. 1974 when expecting a date value) correspond to imprecision. However, although embedded values may contain incomplete information (e.g. "John" when expecting the whole name) they are very hard to quantify and are not considered as lacks of precision. Table 2.15 shows the relation among accuracy factors and error types, indicating the valid (✓) and invalid (×) combinations.

Accuracy factor / *Error type*	**Semantic correctness**	**Syntactic correctness**	**Precision**
Value errors	×	✓	✓
Standardization errors	×	✓	✓
Embedded values	×	✓	×
Wrong values	✓	×	×

Table 2.15 – Relation among accuracy factors and typology of errors

There is also a relation among the *data types* and the *typology of errors*, specifying the kind of errors that can appear in each data type. For example, for some special cases of string data (as addresses) embedded values are frequent. Specific techniques for error detection and correction have been developed for specific data types and specific error types. But note that the relationship between these dimensions is merely syntactical; the correspondence with real world is independent of data types, and consequently, wrong values (values that do not match real world situations) are not related to specific data types. Table 2.16 shows the interrelations among these dimensions categories, indicating the valid combinations; value and standardization errors can appear for all data types, however, embedded values are most frequent for the string type.

Architectural techniques are related to error types. Specifically, the evaluation of wrong values generally implies the comparison with real world, which is generally very costly (in time); therefore, assessment techniques only can be executed when data is materialized. The checking of embedded values also requires expensive routines. Virtual and caching techniques only can execute simple assessment functions, looking for value and standardization errors. Table 2.17 shows the interrelations among these dimensions categories, indicating the valid combinations; value and standardization errors can appear for all types of techniques, however, embedded and wrong values are treated only by materialized techniques.

Data type / *Error type*	**Numeric**	**Date**	**Enumeration**	**String**
Value errors	✓	✓	✓	✓
Standardization errors	✓	✓	✓	✓
Embedded values	×	×	×	✓
Wrong values	no matter data type			

Table 2.16 – Relation among typology of errors and data types

Architectural techniques / *Error type*	**Virtual**	**Caching**	**Materialization**
Value errors	✓	✓	✓
Standardization errors	✓	✓	✓
Embedded values	×	×	✓
Wrong values	×	×	✓

Table 2.17 – Relation among typology of errors and architectural techniques

A priori, all combinations of *granularities of measurement* and *error types* are possible, i.e. cell, set or relationship granularity can be used in the evaluation of the different types of errors (with their valid combinations of data types and architectural techniques).

In addition, *granularity of measurement* is also orthogonal to the accuracy factors, i.e. users may be interested in any accuracy definition, independently of the way accuracy measures are stored. But taking into account the particularities of the systems and the availability of metadata and estimations, the different metrics are generally more related to some levels of granularity. Concretely, semantic correctness ratio and syntactic correctness ratio metrics are not appropriate for cell granularity and scale metric (precision) have no sense for string data types. In addition, as argued in Sub-section 3.2 the standard error metric (precision) is rarely used in DIS applications. Table 2.18 shows the correlation among all taxonomy dimensions and accuracy factors (some metrics are marked in brackets when not all the metrics are appropriate).

Granularity / *Error type – Data type – Arch. techniques*			**Cell granularity**	**Set granularity**	**Relationship granularity**
Value errors	**Numeric Date Enumeration String**	**Virtual Caching Materialization**	Syntactic correctness (deviation) Precision	Syntactic correctness Precision	Syntactic correctness Precision
Standard. errors	**Numeric Date Enumeration String**	**Virtual Caching Materialization**	Syntactic correctness (deviation) Precision (scale, granularity)	Syntactic correctness Precision (scale, granularity)	Syntactic correctness Precision (scale, granularity)
Embedded values	**String**	**Materialization**	Syntactic correctness (deviation) Precision (granularity)	Syntactic correctness Precision (granularity)	Syntactic correctness Precision (granularity)
Wrong values	**No matter data type**	**Materialization**	Semantic correctness (degree, deviation)	Semantic correctness	Semantic correctness

Table 2.18 – Correlation among taxonomy dimensions

The technical problems to solve for each cell of the taxonomy are quite different. For example, measuring semantic correctness at cell granularity requires costly evaluation processes, sometimes requiring human interaction, while counting the ratio of cells do not satisfying a syntactic rule can be easily implemented and executed. In Sub-section 3.6 we discuss evaluation problems.

3.5. Some systems that consider data accuracy

In this sub-section we analyze several types of systems that evaluate accuracy and we describe the goals and problems that they present. We do not try to be exhaustive because there exist lots of propositions that take into account data accuracy; we intend to present an overview of the problems. At the end of the sub-section, Table 2.19 summarizes the discussed proposals in terms of the taxonomy presented before.

Data Warehousing systems

In data warehousing systems, accuracy is studied in the context of data cleaning techniques. The main objective is to detect and correct errors. There is a great variety of techniques for detecting specific errors, generally specialized for some application domains. The most used ones consists in format parsing [Raman+2001], outliers detection [Maletic+2000] and frequency distribution analysis [Quass+1999]. In [Oliveira+2005], authors present a taxonomy of data quality errors (that refines the categories presented in Sub-section 3.3) and some methods (based on decision trees) for deciding which types of errors may arise to data. Regarding error correction, most automatic correction techniques consist in applying user defined functions [Lee+2000] [Galhardas+2000] [Sattler+2000] or simply deleting erroneous values [Vassiliadis+2001] [Sattler+2000]. When errors cannot be automatically corrected, they are reported in a log [Vassiliadis+2001] or interactively presented to the user [Raman+2001] [Galhardas+2000] in order to be manually corrected.

Several cleaning tools have been proposed in the literature, e.g. AJAX [Galhardas+2000], FraQL [Sattler+2000], Potter's Wheel [Raman+2001], ARKTOS [Vassiliadis+2001] and IntelliClean [Lee+2000]. An overview and comparison of tools and the underlying techniques can be found in [Müller+2003] [Oliveira+2004]. The tools bring support for detecting and correcting a wide range of errors, specifically value errors (basically detecting illegal formats and replacing with some derived value), standardization errors (mapping abbreviations and formats to the standard ones), embedded values (looking for frequent patterns and breaking into separate attributes) and missing values (filling-in with derived values). Tools also provide support for detecting and correcting consistency problems, however, they do not treat wrong values (corresponding to semantic correctness problems).

But cleaning techniques do not deal with measuring accuracy levels and informing them to users. Only few works focus on reporting data quality to users. In [Moura+2004], authors build multidimensional cubes to show aggregations of data warehouse quality, some of the dimensions corresponding to indicators related with data accuracy.

In [Zhu+2002] the quality of external sources is evaluated in order to select the most appropriate sources. They study and compare different methods for building quality indicators that aggregate several quality measures; the *syntactic correctness ratio* is one of the proposed quality metrics.

Mediation systems

In mediation systems, *syntactic correctness* is used as a quality metric to compare among sources and to filter the data returned to the user. In [Naumann+1999], they introduce quality factors in a mediation system, which include *syntactic correctness*. They study how to propagate a set of quality factors from several heterogeneous sources to the mediator. The propagation consists basically of merge functions that combine actual values of two sources through a relational operator. They use the *syntactic correctness ratio* metric with a relation/view granularity; considered errors are value errors (misspellings, out-of-range values, etc.).

Cooperative systems

A Cooperative Information System (CIS) is a large scale information system that interconnects various systems of different and autonomous organizations, geographically distributed and sharing common objectives [Mecella+2002]. A service-based framework and an overall architecture for managing data quality in CIS is proposed in [Mecella+2002]. Their architecture allows each organization to export data with associated quality information and to retrieve data specifying quality requirements. Their quality model includes, among other quality factors, the semantic correctness factor, measured as a value-deviation. Quality values are associated to each cell (data is represented in XML documents; quality values follows the same structure). In [Fugini+2002], authors evaluate semantic correctness and syntactic correctness (value errors), both measured as value-deviations

and associated to cells. They use quality measures to support the exchange of trusted data, enabling organizations to assess the suitability of data before using it.

A model for quality management in CIS is presented in [Missier+2001]. They propose an extensive use and exchange of quality metadata. They measure different accuracy factors: semantic correctness, syntactic correctness (value and standardization errors) and precision (scale and granularity). They store the validation history of each cell (validating an item value means testing it, using either some external reference data or some checking algorithm) and discuss its use for data quality enforcement. They use the model in a case of study in the domain of public administration [Missier+2003]. Source data presents a number of problems, e.g. addresses become stale (and then no correspond with real-world), different address formats, same names with multiple spellings and typing errors (generally misspellings). They study assessment methods for their data, classify errors and recommend cleaning techniques for correcting major errors (other errors are lead to manual correction, but their number is substantially reduced).

Other applications

Other types of applications, specifically those managing customers' data, need precise information of the accuracy of each data item. For example, CRM applications manage customer relationship information, generally accessing to multiples sources in order to collect all relevant information and to abstract a customer profile; external sources are increasingly being incorporated. In these applications, business and advertisement information is frequently sent to customers, so having accurate contact information is a major goal. Other information, useful for classifying customers (e.g. profession, income level and preferences), should be accurate enough in order to define good targets for promotions and advertising.

Information quality and particularly information accuracy is indispensable in such applications [Laboisse 2005] [Amat+2005] [Graveleau 2005]. The major goal is achieving semantic correctness. They use a great variety of techniques for detecting and correcting errors, sometimes checking for syntactic correctness in order to find

Works	Measurement	Granularity measurem.	Typology of errors	Data type	Arch. techniques
Data cleaning [Galhardas+2000] [Sattler+2000][Lee+2000] [Raman+2001] [Vassiliadis+2001]	Syntactic correctness	Cell	Value errors, standard. errors, embed. values	All	Materialization
Reporting warehouse quality [Moura+2004]	Syntactic correctness, precision	Set	Value errors, standard. errors, embed. values	All	Materialization
Source selection [Zhu+2002]	Syntactic correctness (ratio)	Set	Value errors	Not specified	Materialization
Quality-based integration [Naumann+1999]	Syntactic correctness (ratio)	Set, relationship	Value errors	Not specified	Virtual
Retrieving quality data in cooperative systems [Mecella+2002]	Semantic correctness (deviation)	Cell	Wrong values	Not specified	Materialization
Trusted data exchange in cooperative systems [Fugini+2002]	Semantic correctness (deviation), syntactic correctness(deviation)	Cell	Value errors, wrong values	Not specified	Materialization
Exchange of quality metadata [Missier+2001] [Missier+2003]	Semantic correctness, syntactic correctness, precision (scale, granularity)	Cell	Value errors, standard. errors, wrong values	All	Materialization
Data cleaning in CRM [Laboisse 2005] [Amat+2005]	Semantic correctness, syntactic correctness, precision (scale)	Cell	All	All	Materialization

Table 2.19 – Summary of proposals

possible semantic errors (e.g. an email that is syntactically incorrect will fail to be delivered). Reference dictionaries (e.g. address dictionaries) are used to verify the existence of data values. But most of the effort corresponds to expensive manual verification tasks, as telephone calls and emails for verifying the exactitude of customer data. In order to involucrate customers in the process (motivate them to answer questions) special promotions and prix are created, further increasing the quality enforcement process. Furthermore, verification and cleaning techniques are often delegated to third party organizations specialized in quality control [Laboisse 2005] [Amat+2005].

3.6. Research problems

Data accuracy has been largely studied for various types of DIS, however, many problems remain unsolved or insufficiently treated. This sub-section summarizes these problems and mentions, when they exist, the references which have done significant contributions in each class of problem. Most of the problems coincide with those ones studied in Sub-section 2.6 for data freshness, so we give only a brief description of the problems and we remit to Sub-section 2.6 for details.

The accuracy evaluation process needs to know about the users' and source profiles, i.e. metadata about users' expectations and source properties. Such elements should be adequately combined in order to evaluate the accuracy of data conveyed to users. Accuracy evaluation can be used for auditing an existing DIS or for designing a new DIS under quality constraints. In the following, we discuss these problems.

Defining users' and source profiles

Evaluating data accuracy implies testing whether user's accuracy expectations can be satisfied from source data accuracy. One of the first problems is how and where to specify user expectations and how to define data source properties which impact data accuracy.

Specification of accuracy expectations

Users have accuracy expectations for their applications which should be formulated according to some factors and metrics among those we have seen in Sub-sections 3.1 and 3.2. The specification of these factors and metrics pose a number of questions:

- Which language or formalism to use? Alternatives can vary from the simple specification of ratios [Kon+1995] [Naumann+1999] [Li+2003a] [Redman 1996] associated to each object type, to the definition of a specialized language if a preference order is introduced among accuracy of different object types.
- At what level accuracy expectations should be specified? Analogously to data freshness, we distinguish four levels: (i) for the whole system, (ii) for each data source, (iii) for each user or group of users, and (iv) for each user query. Technical problems are similar.

Acquisition of source accuracy

The evaluation of data accuracy at source level implies the selection of an accuracy factor and the definition of metrics and measurement processes for it. The definition of new factors and metrics is an interesting area. Most existing works concentrate in the syntactic correctness factor, while industry is increasingly demanding better techniques for evaluating semantic correctness and precision. Furthermore, the current development of third party service societies specialized in the hosting and measurement of accuracy of enterprise data confirms its importance [Laboisse 2005]. The acquisition of source data accuracy poses some questions:

- Which source metadata is necessary to represent accuracy in a source profile? In order to characterize source actual accuracy some metadata should be obtained, for example error distribution. This metadata will depend on the level of granularity (cell, set or relationship) choose for expressing accuracy metrics.
- How to acquire such metadata from sources? Some sources (or domain experts) can provide useful information as the confidence degree for certain attributes, but for other sources these values must be

learned or estimated from statistics elaborated during the data source exploitation. There is a large collection of techniques and functions for measuring some types of errors, commonly known as error detection techniques. The most used ones consists in format parsing [Raman+2001], outliers detection [Maletic+2000] and frequency distribution analysis [Quass+1999]. Several cleaning tools have been also proposed, e.g. AJAX [Galhardas+2000], FraQL [Sattler+2000], Potter's Wheel [Raman+2001], ARKTOS [Vassiliadis+2001] and IntelliClean [Lee+2000]. Both techniques and tools bring support for detecting a wide range of syntactical errors (value errors, standardization errors and embedded values) but do not treat wrong values (corresponding to semantic correctness problems). Techniques for specific metadata should be also developed as in [Laboisse+2005].

Auditing data accuracy

Having users and source profiles, one of the challenging problems is how to combine these elements in order to evaluate the accuracy of the data conveyed to users. The main questions are: How should the different parameters of the source profile be combined for evaluating data accuracy? What is the impact of error distribution in the accuracy of data?

An additional question is how to combine several source actual values to obtain a global quality estimation. There is a proposal for combining accuracy ratios within SQL operators, following gross hypothesis of uniform distribution of errors [Naumann+1999]. The combination function is very simple (product of the input values). Error models that better represent source data should be analyzed. An important contribution in this line was presented in [Motro+1998], partitioning source data according to error distribution. The impact of complex operations such as data cleaning processes are not treated at all.

Accuracy evaluation techniques can be used in the development of auditing tools, responsible for the evaluation of data quality. Several kinds of auditing tools can be conceived, for example:

- Prediction tools, for predicting the quality of data that can be returned in response to a query, without executing the query.
- Integrated evaluation tools, for measuring data quality during query execution and labeling delivered data with its quality levels. These tools can be integrated to the query evaluation process.
- Statistical tools, for taking samples of data quality during query execution and storing statistics. These tools can serve the first two categories.

The tools should use metadata describing the sources, the DIS and user expectations. They can be used at design time to evaluate the system, for example to test if user expectations can be achieved, or they can be used at run time for example, to predict the quality of the data delivered by alternative processes in order to choose the process that best suites user quality expectations.

Quality-driven engineering

Quality-driven engineering consists in designing a DIS under quality constraints. This kind of techniques may help in the selection among several design choices:

- Selection among alternative query plans that access alternative data sources.
- Specific error correction techniques as duplicate elimination.
- Relaxation of selection or join conditions, replacing crisp predicate for approximate ones.

Although several kinds of techniques have been explored in depth for specific systems, adapting their capabilities to heterogeneous systems requires addressing additional problems, as the combination of source data with different quality values, or even the comparison of data source profiles.

The challenge in quality-driven engineering is in the identification of the variables that influence data accuracy and the proposition of techniques to achieve such improvements. Most existing work consists in automating error detection and correction, either applying user defined functions [Lee+2000] [Galhardas+2000] [Sattler+2000] or simply deleting erroneous values [Vassiliadis+2001] [Sattler+2000]. When errors cannot be automatically corrected, they are reported in a log [Vassiliadis+2001] or interactively presented to the user [Raman+2001] [Galhardas+2000] in order to be manually corrected. Existing cleaning tools bring support for detecting and correcting a wide range of errors, specifically value errors (basically detecting illegal formats and

replacing with some derived value), standardization errors (mapping abbreviations and formats to the standard ones) and embedded values (looking for frequent patterns and breaking into separate attributes). However, they do not treat wrong values (corresponding to semantic correctness problems).

Certain works propose the selection of pertinent sources according to their quality [Zhu+2002] [Mihaila+2000] [Naumann+1998]. In particular, data accuracy is taken into account in [Zhu+2002]. They compare different methods for building quality indicators that aggregate several quality measures, in order to rank sources according to such indicators. None of the proposals takes into account user expectations.

As improving accuracy is not the unique quality goal for a given system, the relationship with other quality properties becomes a major challenge in DIS design. The relationship among accuracy and other quality properties has been only partially studied. There are two main lines: (i) tuning other properties in order to optimize accuracy, and (ii) relaxing accuracy in order to optimize other quality factors. In the former, the challenge is in the identification of related quality properties and the proposition of techniques that take advantages from these relationships in order to improve the global quality of data. Existing works in this line mainly concern the balance of semantic correctness and timeliness [Ballou+1995], syntactic accuracy and completeness [Ballou+2003], semantic correctness, completeness and timeliness [Cappiello+2002] and semantic correctness, process timeliness and cost [Han+2003][*]. The latter is not treated in the literature.

Discussion

Among the discussed research problems, this thesis deals with data accuracy auditing and accuracy-driven engineering. Regarding data accuracy auditing, we are interested in the combination of relevant parameters (source data accuracy, error distribution, user expectations) in order to evaluate the accuracy of data conveyed to users. We fix our context to the relational model and we take into account how inaccuracies are distributed in source relations, inspired in the partitioning mechanism proposed in [Motro+1998]. Data accuracy values are used in an accuracy-driven application which goal is to answer user queries using data that verifies accuracy expectations. Contrarily to source selection proposals [Zhu+2002] [Mihaila+2000] [Naumann+1998] which select whole sources relations (or whole sources), we consider that source relations are partitioned according to data accuracy and we answer user queries using the areas of source relations that better adapt to user needs. Chapter 4 deals with data accuracy evaluation and enforcement.

4. Conclusion

In this chapter we described two of the most used quality dimensions: data freshness and data accuracy. We analyzed several factors and metrics proposed in the literature to measure them and we explored the dimensions that influence their evaluation, which were organized in taxonomies. Guided by the taxonomies, we classified some works that consider data freshness and data accuracy and we analyzed open research problems. Both analyzes, for data freshness and for data accuracy, shown that existing work concentrates for some specific types of DIS, specific characteristics (e.g. materialized data, some types of errors) or specific metrics but other configurations remain untreated.

As research problems we identified the specification of user expectations, acquisition of source data quality, formulation of cost models, data quality auditing and quality-driven engineering. Among these problems, we are interested in the two latter. Specifically, this thesis deals with the evaluation of freshness and accuracy of the data conveyed to users in DIS applications. The obtained quality values are analyzed and compared to user expectations (i.e. quality auditing) and are input for specific techniques for quality improvement (i.e. quality-driven engineering). The following chapters develop the proposal.

[*] Semantic correctness is called accuracy in [Ballou+1995], [Cappiello+2002] and [Han+2003]; syntactic accuracy (specifically the conformance to a standard) is called consistency in [Ballou+2003] and timeliness is called currency in [Cappiello+2002].

Chapter 3. Data Freshness

This chapter describes our proposal for data freshness evaluation and enforcement. We propose freshness evaluation algorithms that take into account the properties of the data integration system that have more impact in data freshness. This approach allows the specialization of evaluation algorithms according to different application scenarios. We also present an approach for data freshness enforcement when freshness requirements cannot be achieved.

1. Introduction

The needs of having precise measures of data freshness become increasingly critical in several fields. Examples are numerous:

- *Information Retrieval*: Given a user query, there may be a great number of web sources providing data to answer the query but having different data quality, in particular, having varied freshness. A big amount of retrieved data is not relevant for users because of its lacks of freshness (e.g. web pages announcing train ticket reductions for expired promotions). The analysis of data freshness is useful for making a pre-filtering of data (or entire data sources), according to freshness requirements. In addition, retrieved data may be sorted according to their freshness, allowing the user to first see the freshest data.
- *Decision making*: When decision making is based on data extracted from autonomous data sources, external to the organization, a fine knowledge of data quality is necessary in order to associate relative importance to data. For example, old euro currencies should not impact decisions in the same way than more recent currencies. In this context, data freshness should be informed to end-users, as an additional attribute qualifying data. Further strategies, as filtering old data may also be carried out.
- *E-commerce*: Many web portals, such as Kelkoo® or Ebay® bring a uniform access to products of several vendors, allowing the comparison of product features and prices. Offerings are so numerous and disparate that users are overloaded with great amounts of data. Frequently, many of the proposed products are out of stock or have changed the offering conditions (e.g. the price); for example, when searching last minute flights, most of the offers are timeout and user loses considerable time. Incorporating data freshness conditions to the query interfaces offers a good possibility for reducing interaction times and providing relevant data.
- *Scientific experiments*: Research experiments, especially in the field of life sciences, produce great amounts of data, which are published in databanks and journals. Searches of related experiments (e.g. about a gene sub-sequence) are frequently carried out in order to cross results and abstract similar behaviors. Comparison is not trivial and requires executing expensive routines, which is worsen by the great amount of data and its wide overlapping. Furthermore, the existence of relatively old data introduces noise to the task. In this context, the analysis of data freshness may help reducing the search space in order to retrieve the most recent experiments.
- *Web-services integration*: Consider a company that uses web service S and wants to find a compatible service provider. The selection among the offered services may be done according to several criteria, for example, response time or service availability. When the service also provides data, data freshness may play an important role, because users may prefer obtaining the most recent data. For example, in an application querying yellow pages providers, data freshness may be critical.
- *Customer relationship management*: Managing obsolete data may become very expensive. A well-known example is the return of mail because customers have changed their addresses while the system continues managing the old ones. Generally, many data qualifying customers are obtained from external sources (e.g. address catalogs, yellow pages, census data). Knowing data freshness is crucial for taking accurate decisions. Furthermore, data freshness may be an important factor when choosing among data providers.

All these scenarios motivate the need of data freshness evaluation methods capable of adapting to different user expectations and different perceptions of data freshness. As argued in previous chapter, data freshness represents a family of quality factors. We recall the two freshness factors that have been proposed in the literature (see Sub-section 2.1 of Chapter 2 for further details):

- *Currency* describes how *stale* is data with respect to the sources. It captures the gap between the extraction of data from the sources and its delivery to the users. It is often measured as the time elapsed since data was extracted from the source.
- *Timeliness* describes how *old* is data (since its creation/update at the sources). It captures the gap between data creation/update and data delivery no matter when data was extracted from sources. It is often estimated as the time elapsed from the last update to a source.

We consider both freshness factors. We use the term *data freshness* when the discussion concerns both factors and we refer to *data currency* and *data timelines* only when specific discussion is necessary. We consider set granularity for measures, i.e. a freshness value is associated to each source relation (or equivalent structures when sources are not relational).

In this chapter we deal with data freshness evaluation in data integration systems (DISs). We address the problem of evaluating the freshness of the data returned to users in response to their queries and deciding if users' freshness expectations can be achieved. Initially, we treat the topic of data freshness evaluation, and then, we discuss how freshness measures can be used for improving the DIS and enforcing data freshness.

In order to evaluate the freshness of the data returned to users, we should consider the freshness of source data and also take into account the processes that extract, integrate and convey data to users. In previous chapter we analyzed the various dimensions that influence data freshness, namely, nature of data, architectural techniques and synchronization policies. We now focus on modeling such features and using them in the freshness evaluation process. To this end, we propose a framework which attempts to formalize the different elements involved in data freshness evaluation. Among these elements there are data sources, user queries, processes that extract, integrate and convey data, metadata describing DISs features, quality measures and quality evaluation algorithms.

In our framework, DISs are modeled as workflow processes in which the workflow activities perform the different tasks that extract, integrate and convey data to end-users. For example, in data warehouse refreshment processes, typical workflow activities are the routines that perform the extraction, cleaning, integration, aggregation and customization of data [Bouzeghoub+1999]. Workflow models enable the representation of complex data manipulation operations. Quality evaluation algorithms are based on the workflow's graph representation and consequently, the freshness evaluation problem turns into value aggregation and propagation through this graph.

The idea behind the framework is to define 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. We propose a freshness evaluation approach that is general enough to be used in different types of DISs but is flexible enough to adapt to the characteristics of concrete application scenarios.

In addition to allowing the evaluation of data freshness, our framework proposes many facilities for data freshness enforcement. A DIS should provide the data freshness expected by the users. In order to know if user freshness expectations can be achieved by the DIS, we can evaluate the freshness values of conveyed data and compare them with those expected by users. If freshness expectations are not achieved, we may improve DIS design in order to enforce freshness or negotiate with source data providers or users in order to relax constraints. We propose a freshness enforcement approach that supports the analysis of the DIS at different abstraction levels in order to identify its critical points and to target the study of improvement actions for these critical points.

The following sections describe our approach for data freshness evaluation and enforcement: Section 2 describes the framework and presents an overview of the evaluation approach. Section 3 uses the framework for data freshness evaluation, specifically, we model the DISs processes and properties that have impact in data freshness and we implement evaluation algorithms that take into account the processes and properties. Section 4 deals with data freshness enforcement, presenting improvement actions for enforcing data freshness when freshness expectations cannot be achieved by a DIS. Section 5 illustrates the development of a specific improvement action. We conclude, in Section 6, by drawing the lessons learned from our experiments.

2. Data quality evaluation framework

In this section we present a framework for data freshness evaluation in the context of DISs. The framework models data sources, data targets and the DIS processes (which build target data from source data). DIS processes include the tasks for extracting, transforming and integrating data and conveying it to users. We can model a unique DIS or several DISs (e.g. various data marts for different departments of an enterprise).

The goal of the framework is twofold, firstly, helping in the identification of the DIS properties that should be taken into account for freshness evaluation, and secondly, allowing the easy development of evaluation algorithms that consider such properties.

This section describes the quality evaluation framework, specifying the representation of its components and presenting an overview of its usage for data quality evaluation. The rest of the chapter provides detailed description on data freshness evaluation and enforcement, based on this framework.

2.1. Definition of the framework

The quality evaluation framework attempts to formalize the different elements involved in data freshness evaluation. Among these elements there are data sources, data targets, DIS processes, DIS features, quality measures and quality evaluation algorithms. We start defining the framework and along the section we define the framework components.

The proposed framework is defined as follows:

Definition 3.1 (quality evaluation framework). The *quality evaluation framework* is a 5-uple:

<Sources, Targets, QualityGraphs, Properties, Algorithms>

where *Sources* is a set of available data sources, *Targets* is a set of data targets, *QualityGraphs* is a set of graphs representing several DISs processes, *Properties* is a set of properties describing DISs features and quality measures and *Algorithms* is a set of quality evaluation algorithms. □

Example 3.1. Consider a DIS that retrieves meteorological information from three sources: S_1 (real time meteorological data of satellites), S_2 (meteorological dissemination database) and S_3 (climatic sensors). The DIS provides information to three query interfaces: T_1 (historical information about climate alerts), T_2 (aggregated data about climate measurements) and T_3 (detailed data about predictions). The DIS includes processes for extracting, filtering, integrating and aggregating data. Among the DIS features that are relevant for studying data freshness there are, for example, the processing cost of DIS processes and the refreshment frequencies of materialized data. As quality measure, users are interested in data timeliness. Consequently, there is a quality evaluation algorithm for measuring data timeliness in such DIS. The quality evaluation framework allows modeling all these components. □

The framework includes a set of data targets for which the user requires data quality evaluation and a set of data sources providing data for feeding those data targets. Data sources can be relations in data repositories, web pages, user input interfaces or other types of applications producing data. Analogously, data targets can be relations in data repositories, views, user display interfaces or other types of applications consuming data. Sources and targets are defined as follows:

Definition 3.2 (data source). 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 3.3 (data target). 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). □

Each DIS extracts data from some data sources and provides some data targets with data. Those sources and targets are included in the sets of sources and targets of the framework. Next sub-section describes a model for DISs processes.

2.1.1. Graph model of the data integration system workflow

A 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. Each workflow activity takes data from sources or other activities and produces result data that can be used as input for other activities. Then, data traverses a path from sources to users where it is transformed and processed according to the system logics. We choose workflow models in order to enable the representation of complex data manipulation operations, as in [Bouzeghoub+1999] [Grigori+2005] [Ballou+1998]. In order to perform data quality evaluation, we define 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. Many of existing proposals for workflow specification are graph based [van der Aalst+2002] [Mendling+2006] [Ziemann+2005] [Grigori+2005] and many works have represented DIS as graphs [Theodoratos+1997] [Naumann+1999]. For this reason, we choose to base our quality evaluation approach on graphs*. Using graphs as representation formalism, the quality evaluation problem turns into a graph traversal problem.

A quality graph is a directed acyclic graph. The nodes are of three types: (i) *activity nodes* representing the major tasks of a DIS, (ii) *source nodes* representing data sources accessed by the DIS, and (iii) *target nodes* representing data targets fed by the DIS. Nodes have a name that identifies them; for source and target nodes, their name coincide with those of sources and targets of the framework. Activities can be atomic or composed. They consume input data elements and produce output data elements which may persist in repositories. There are two types of edges: (i) *control edges* expressing the control flow dependencies between activities (e.g. execution precedence), and (ii) *data edges* representing data flow from sources to activities, from activities to targets and between activities (i.e. the output data of an activity is taken as input by a successor activity). In most DISs, control flow is induced by data flow, i.e. there is a control flow edge between two activities if and only if there is a data flow edge between them. Both, nodes and edges can have labels, which are discussed in next sub-section.

In summary, a quality graph is defined as follows:

Definition 3.4 (quality graph). A *quality graph* is a quadruple $G=(V, E, \rho_V, \rho_E)$ 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 $u,v \in V$, $t \in T$.
- $\rho_V : V \rightarrow L_V$ is a function assigning labels to the nodes. L_V denotes the set of node labels.
- $\rho_E : E \rightarrow L_E$ is a function assigning labels to the edges. Analogously, L_E denotes the set of edge labels. □

We consider, without loss of generality, that target nodes have a unique incoming data edge. If we need to model multiple data edges incoming a target node, we can add a virtual activity node that concentrates the incoming edges and has a unique outgoing edge to the target node. This pattern is commonly used in workflow design [van der Aalst+2003]. Analogously, we consider that source nodes have a unique outgoing data edge.

Figure 3.1 sketches the quality graph representation. Activity nodes are represented as circles, while source nodes (with no input edges) and target nodes (with no output edges) are represented as rectangles. Data edges are continuous arrows and control edges are dotted arrows. Labels are written next to nodes and edges. In some figures, when we only want to study the data flow, control edges may be omitted. Analogously, some properties may be omitted.

* We use graphs for representing workflows although the approach can easily be adapted to other formal models such as Petri nets or state-chart diagrams, provided that it is a uniform model.

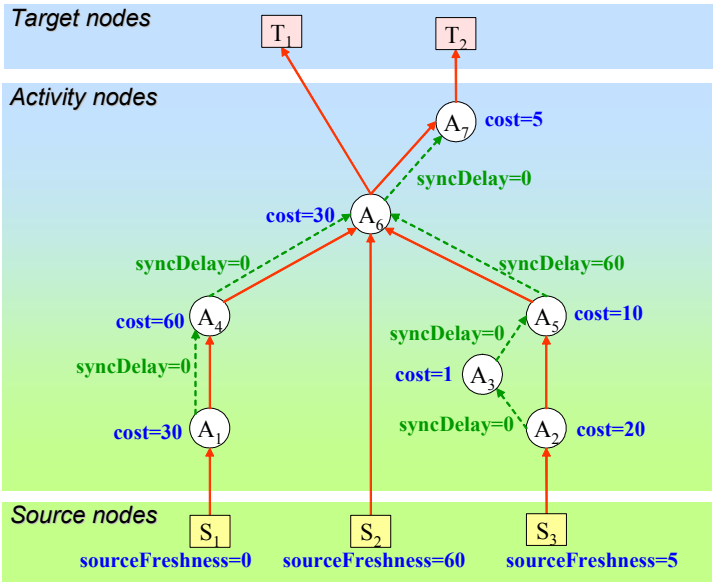

Figure 3.1 – Quality graph

2.1.2. Quality features as graph adornments

Data quality depends on the structure of the quality graph (i.e. how data traverses the graph) but also on data and process properties (e.g. process synchronization policies, data volatility). The idea behind this approach is to adorn quality graphs 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.

Properties can be of two types: (i) *features*, indicating some characteristic of the DIS (costs, delays, policies, strategies, constraints, etc.), or (ii) *measures*, indicating a quality value corresponding to a quality factor.

Features can represent precise process metadata (as execution cost of activities), estimations (e.g. based on designer experience on similar applications, cost models or upper bounds) or beliefs (e.g. source reputation). Quality measures can be actual values acquired from sources (e.g. source data freshness, measured from data last update) or expected values indicating user high level expectations (as the desired data freshness). Property values (for both, features and measures) can be directly given by system administrators, users or source providers (e.g. source availability windows, DIS refreshment policies, user expected freshness), can be systematically obtained using measurement processes (e.g. response time), can be aggregated from user passed behavior or statistics (e.g. user preferred sources, activity execution time), can be derived from other property values (e.g. activity global cost) or can be calculated in some ad-hoc way. Quality evaluation algorithms calculate further property values (quality measures) as will be discussed in next sub-section.

We define a property as follows:

> **Definition 3.5 (property).** A *property* is a 3-uple <name, metric, domain> where *name* is a String that identifies the property, *metric* is a description of the measurement semantics and units, and *domain* describes the domain of the property values. □

Nodes and edges of quality graphs are adorned with property labels of the form: property = value, where *property* is a property name and *value* is an element of the property domain. In Figure 3.1, property labels (cost, synchronization delay and source freshness) are written next to nodes and edges.

In the following sub-section we illustrate how we utilize property labels for evaluating data quality.

2.2. The approach for data quality evaluation in data integration systems

Quality evaluation is performed by evaluation algorithms that take as input a quality graph and calculate the quality values (corresponding to data freshness) for the graph. In order to illustrate our quality evaluation approach, consider the following example:

Example 3.2. Figure 3.2 sketches a quality graph representing a simple DIS, which extracts traffic statistics of a unique data source (S_1). A wrapper (activity A_1) extracts data, which is cleaned and prepared by activity A_2 and finally aggregated (activity A_3) and delivered to a user application (T_1).

We aim to estimate data timeliness. Consider that S_1 is a dissemination server that publishes traffic statistics once an hour, so published statistics correspond to traffic events of the passed hour. A_1 extracts data immediately after its publication. Source data timeliness (at extraction time) is an hour (60 minutes) because the oldest source data may be produced at most an hour before. Also consider that the execution costs of activities A_1, A_2 and A_3 are 5, 60 and 15 minutes respectively, as shown in Figure 3.2. These costs should be taken into account in the estimation of the freshness (timeliness) of the data returned to users (through user application T_1). Intuitively, timeliness of resulting data is estimated as 140 minutes, which results from adding the execution cost of activities to the source data freshness (60 + 5 + 60 + 15). □

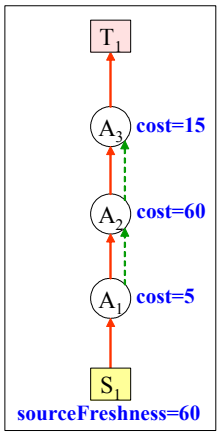

Figure 3.2 – Quality evaluation example

Evaluation algorithms may traverse the graph, node by node, operating with property values. For example, a simple evaluation algorithm may start at source nodes, read the value of the *source freshness* property and move along the data flow adding the value of the *cost* property of each activity, as intuitively explained in previous example. Such algorithm can easily be implemented using default graph traversal methods. This mechanism for calculating data quality applying operations along the graphs is what we call *propagation of quality measures within the graph* (*quality propagation* for short).

As a quality graph describes the DIS integration process and its properties, it contains the input information needed by evaluation algorithms. Evaluation algorithms take as input a quality graph, calculate the quality values corresponding to a quality factor and return a quality graph with an additional property (corresponding to the evaluated quality factor). The quality graph must be labeled with certain property values in order to execute a certain algorithm. For example, the activity nodes of a quality graph must be labeled with their execution cost in order to execute the evaluation algorithm informally described in Example 3.1.

Formally, evaluation algorithms are defined as follows:

Definition 3.6 (quality evaluation algorithm). A *quality evaluation algorithm* is a 7-uple:

<Name, Description, QualityFactor, Input, Output, Preconditions, Postconditions>

where:

- *Name* is a String that identifies the algorithm.
- *Description* is a free-form text describing algorithm evaluation strategy and optional details.
- *QualityFactor* is the quality factor that the algorithm calculates.
- The *Input* is a quality graph.
- The *Output* is a quality graph that results of adding new property values (corresponding to the algorithm *quality factor*) to the *input* quality graph.
- *Preconditions* is a set of pairs <group, property> indicating that a group of nodes/edges (e.g. activity nodes) must be labeled with values of such property (e.g. execution cost). The algorithm can be executed only if preconditions are satisfied.
- *Postconditions* is a set of groups of nodes/edges that will be labeled with new property values (corresponding to the algorithm *quality factor*) after algorithm execution. □

Concerning code, evaluation algorithms have the following signature:

```
FUNCTION AlgorithmName (G: QualityGraph) RETURNS QualityGraph
```

The implementation of evaluation algorithms may vary according to the quality factor and the concrete application scenario. The framework does not constrain the way the algorithms can be implemented. For example, the simple evaluation algorithm of Example 3.1 propagates freshness values adding property values, but another evaluation algorithm may use sophisticated calculation strategies and even user-defined functions. The proposed graph representation facilitates the implementation because it enables to use graph primitives (e.g. getPredecessors, getSuccessors, getProperties) and traversal methods (e.g. findShortestPath, depthFirstSearch).

There are two kinds of quality propagations: (i) propagation of actual values, and (ii) propagation of expected values. In the former (as illustrated in previous example), quality values of source data are propagated along the graph, in the sense of the data flow (from source to target nodes) and combined with property values of nodes and edges. In the latter, quality values expected by users are propagated along the graph but in the opposite sense (from target to source nodes) and combined with property values.

> **Example 3.3.** Consider that users expect freshness values of at most 2 hours (120 minutes) for data produced by the DIS of Figure 3.2. Subtracting activity execution costs from the freshness expected value, we obtain a value of 40 minutes (120 – 15 – 60 – 5). This value means that the source should provide data that is fresher than 40 minutes in order to satisfy user freshness expectations. □

Propagation of actual values serves to inform users of the quality of result data; a comparison with user expectations at target nodes (expected values versus propagated actual values) determines if user quality expectations can be achieved or not. Conversely, propagation of expected values serves to constraint source providers on the quality of source data (or to choose among alternative sources providing the same type of data); a comparison with source actual quality at source nodes (actual values versus propagated expected values) determines if sources provide data with enough quality.

An important remark is that quality propagation can be performed during the execution of DIS processes using precise property values (e.g. exact execution cost obtained during execution), or conversely, it can be performed without executing DIS processes and using estimations of property values (e.g. statistics of costs of previous executions). In the former, quality evaluation is used to inform users of results quality. In the latter, quality estimations can be used to decide whether query results will be adequate for user needs or not (if no adequate, alternative actions may be taken, for example, accessing to sources with higher quality even if source accesses are more expensive). We focus on this latter propagation context, even if our mechanism can be easily applied for calculating data quality during DIS execution.

Finally, note that although the framework was conceived for the evaluation of data freshness, their components were defined in a general way allowing its use for the evaluation of other quality factors. For example, the *response time* quality factor may be measured in a similar way, and the direct use of the framework for its evaluation seems to be possible. In order to evaluate other quality factors, the framework may need to be extended, adding for example, new propagation operations; Chapter 4 illustrates this fact for the data accuracy quality factor.

In the next section we use the framework for data freshness evaluation.

3. Data freshness evaluation

In this section we describe our data freshness evaluation approach. We firstly give an intuitive idea of the freshness calculation strategy and we describe some general properties that support the calculation of data freshness. Then, we present a basic data freshness propagation algorithm based on those properties. Finally, we describe an instantiation approach for adapting the basic algorithm to particular application scenarios.

We firstly explain the propagation of freshness actual values (Sub-sections 3.1 to 3.5); the propagation of freshness expected values is similar and is discussed in Sub-section 3.6. We conclude this section motivating some direct applications of both types of data freshness propagations.

3.1. Basic evaluation algorithm

In this sub-section we propose a basic algorithm for evaluating data freshness. In order to propagate freshness actual values, the algorithm needs input information describing DIS properties.

The freshness of the data delivered to users depends on the freshness of source data but also on the amount of time needed for executing all the activities as well as on the delays that may exist among their executions. We briefly describe such properties, as well as users' freshness expectations:

- *Processing cost*: It is the amount of time that an activity needs for reading input data, executing and building result data (*cost* is used for short in some figures).
- *Inter-process delay*: It is the amount of time passed between the executions of two activities, i.e. between the end of the former and the start of the latter (*delay* for short).
- *Source data actual freshness*: It is the freshness of data in a source, i.e. at the moment of data extraction (*sourceAfreshness* for short). As data currency measures the gap with source data (the time passed since data extraction) data currency of source data is always zero, however, data timeliness of source data can take positive values.
- *Target data expected freshness*: It is the users' desired freshness for result data (*targetEfreshness* for short).

The evaluation algorithm calculates the following property:

- *Actual freshness*: It is an estimation of the actual freshness of data outgoing a node (*Afreshness* for short).

The relevance of these properties depends on the application scenario. For example, the materialization of data and the use of different policies to refresh such data may imply important *inter-process delays* while in virtual systems these delays may be negligible. The calculation (or estimation) of such property values is discussed in Sub-section 3.4, taking into account the particularities of concrete scenarios.

We propose an evaluation algorithm that estimates the freshness of result data based on previous properties. The algorithm propagates freshness actual values traversing the quality graph (following the data flow) and calculating the freshness of the data outgoing each node. The principle is the following:

- For a source node A, the freshness of data outgoing A is calculated as the source data actual freshness.
- 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.
- In the general case, if activity A has several predecessors, the freshness of data coming from each predecessor (plus the corresponding inter-process delay) should be combined (synthesizing a unique value) 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 (Sub-section 3.4.2 discusses other combination functions).

The calculation can be sketched as shown in Figure 3.3: First, for each predecessor ($P_1 \dots P_n$) of activity A, we add the freshness of incoming data and the inter-process delay. Then, we combine such values (combineActualFreshness function) and add the processing cost to the result. The resulting value is associated to the data edges going from A to its successors ($C_1 \dots C_m$).

1. v_1 = Freshness(P_1,A) + InterProcessDelay(P_1,A) …
   v_n = Freshness(P_n,A) + InterProcessDelay(P_n,A)
2. F = combineActualFreshness({v_1,…v_n}) + ProcessingCost(A)
3. Freshness(A, C_1) = F…
   Freshness(A, C_m) = F

Figure 3.3 – Freshness evaluation strategy

We associate freshness values to the data edges outgoing nodes because we want to emphasize that *freshness* is a property of data not of processes. However, the strategy can be very easily adapted for associating freshness

values to nodes. For practical reasons, sometimes we refer to the *freshness of a node*, meaning the *freshness of data produced by the node*, i.e. freshness values associated to outgoing data edges.

Note that in previous formulas (Figure 3.3) we need inter-process delay values to be associated to data edges. However, inter-process delays may be influenced by properties of control flow (e.g. the type of synchronization among activities), which should be taken into account in their calculation. The calculation (or estimation) of property values is discussed in Sub-section 3.4. By the moment, we can consider that property values are labels of the quality graph as in Figure 3.4a. Processing costs are associated to activity nodes and inter-process delays are associated to data edges among activities, however, in order to facilitate the expression of some formulas (e.g. those of Figure 3.3) we often consider that processing costs are associated to all nodes (with zero value for source and target nodes) and that inter-process delays are associated to all data edges (with zero values for edges outgoing source nodes or incoming target nodes).

> **Example 3.4.** Consider the quality graph of Figure 3.4a as input for the propagation of freshness actual values. The freshness of (S_1,A_1) is calculated as the source data actual freshness of S_1, i.e. 10 units of time. As A_1 has a unique predecessor S_1, the freshness of (A_1,A_3) is calculated adding actual freshness of (S_1,A_1) plus the inter-process delay of (S_1,A_1) plus the processing cost of A_1, obtaining a value of 13 (10+0+3) units of time. As A_3 has two predecessors (A_1 and A_2), two input values are combined: 18 (13+5) and 12 (10+2), keeping the maximum, which is added to the processing cost of A3. Then, freshness of (A_3, T_1) is 22 (18+4) units of time. Figure 3.4b shows the output quality graph. □

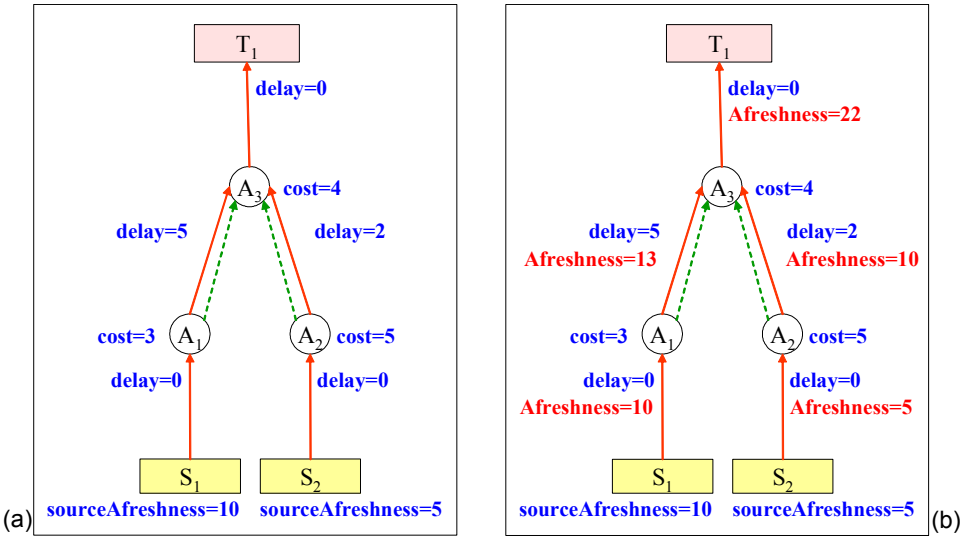

Figure 3.4 – Propagation of freshness actual values: (a) input quality graph, and (b) output quality graph

The previous strategy is implemented in a basic algorithm for freshness propagation: *ActualFreshnessPropagation* (see Algorithm 3.1). It takes as input a quality graph and returns as output the quality graph with additional labels corresponding to data freshness. The *QualityGraph* class has methods for manipulating the classical graph operations (as *getPredecessors*) and for manipulating the property values associated to the nodes and edges (as *addProperty* and *getPropertyValue*). The algorithm first spans source nodes, obtaining source data actual freshness and storing the freshness of data outgoing source nodes. Then, the algorithm traverses activity nodes, obtaining data freshness and inter-process delays of incoming edges and storing such values in a list (valList). The values of the list are combined, added to the processing cost of the activity, and finally stored for all outgoing edges. A pseudocode of the algorithm is sketched in Algorithm 3.1.

We defined the *getSourceActualFreshness*, *getInterProcessDelay* and *getProcessingCost* functions, as abstract functions, which should be instantiated for specific scenarios. The functions calculate the values of the source data actual freshness, inter-process delay and processing cost properties. Their signatures are:

```
FUNCTION getSourceActualFreshness (G: QualityGraph, A: Node) RETURNS INTEGER
FUNCTION getProcessingCost (G: QualityGraph, A: Node) RETURNS INTEGER
FUNCTION getInterProcessDelay (G: QualityGraph, e: Edge) RETURNS INTEGER
```

```
FUNCTION ActualFreshnessPropagation (G: QualityGraph) RETURNS QualityGraph
   INTEGER value;
   FOR EACH source node S of G DO
      value= getSourceActualFreshness(G,S);
      Edge e = data edge outgoing S in G
      G.addProperty(e,"ActualFreshness",value);
   ENDFOR;
   FOR EACH activity node A in topological order of G DO
      HASHTABLE valList;
      FOR EACH data edge e incoming A in G DO
         value= G.getPropertyValue(e,"ActualFreshness") + getInterProcessDelay(G,e);
         valList.add(e,value);
      ENDFOR;
      value= combineActualFreshness(G,valList) + getProcessingCost(G,A);
      FOR EACH data edge e outgoing A in G DO
         G.addProperty(e,"ActualFreshness",value);
      ENDFOR;
   ENDFOR;
   RETURN G;
END
```

Algorithm 3.1 – Basic algorithm for propagating freshness actual values

The *combineActualFreshness* function is also an abstract function that combines the freshness values of predecessor nodes (stored in the *valList* array, including the delays) and synthesize a unique freshness value. For example, if users are interested in an upper bound of freshness the *combineActualFreshness* function may return the maximum of predecessors freshness. Its signature is:

```
FUNCTION combineActualFreshness (G: QualityGraph, valList: HashTable) RETURNS INTEGER
```

The previous abstract functions can be overloaded for different scenarios taking into account the characteristics of the scenarios. Next sub-sections discuss these ideas.

3.2. Overview of the instantiation approach

In order to characterize different types of scenarios we consider the freshness factor that users are interested in and the three dimensions that influence data freshness (introduced in Chapter 2 - Sub-section 2.3), namely, nature of data, architectural techniques and synchronization policies. Our goal is to develop data freshness evaluation algorithms specialized for the different scenarios.

The basic evaluation algorithm can be specialized for considering the particularities of the scenarios. Firstly, different DIS properties should be considered in the evaluation. These properties are the ones that allow the estimation of source data actual freshness, processing costs and inter-process delays for a given scenario. For example, when materializing data, the time passed between two consecutive refreshments (refreshment period) may be an important component of the *inter-process delay* while in virtual systems this property has no sense. In addition, the evaluation algorithm should be instantiated to take into account the considered properties. The instantiation consists in overloading the abstract functions according to the scenario properties.

Then, the instantiation method consists of three steps:

1. Modeling the scenario according to the freshness factors and the dimensions that influence data freshness.
2. Identifying the appropriate properties for the scenario.
3. Instantiating the evaluation algorithm.

In order to illustrate the instantiation approach, we introduce the following motivating example:

Example 3.5. Consider three different DIS scenarios that deal with information about cinemas and films, illustrated in Figure 3.5:

- **DIS$_1$**: A DIS that retrieves information about films and the cinemas where these films are in billboard, in response to user queries as "Where can I see a given film?" or "Which films are in billboard now?". It extracts film information (titles, genre, actors, directors, timetables, etc.) from the AlloCiné site and cinema information (cinema, capacity, category, etc.) from the UGC and CinéCité sites. The process model consists of three activities for extracting data from the mentioned sites (A_1, A_2 and A_3), an activity for merging (union) the data extracted from both cinema sites (A_4) and an activity for joining film and cinema information (A_5). Users expect data freshness of at least a week.
- **DIS$_2$**: A DIS (part of a reservation system) that accesses to information about cinemas and the availability of places for their performances (of the UGC and CinéCité sites) and present the information to the user allowing him to choose a cinema. The process model consists of two activities for extracting data from both sites (B_1 and B_2) and an activity (B_3) that merges the extracted data, formats it and conveys it to the user interface. When activity B_3 receives data from an extractor, it waits for data from the other extractor for at most one minute, conveying the available data to the user. User freshness expectations are of at least 5 minutes.
- **DIS$_3$**: A DIS that manages statistics information about films, the number of persons that watched each film and their opinions. Typical user questions are "Which films have the best ranking this week?" or "Which film should I watch?". The DIS extracts film information and audience statistics from the AlloCiné site and critic information (films, opinions, recommendations, etc.) from the CineCritics and FilmCritiquer sites. Such sources are queried at different times (film data is extracted weekly and critic data daily) because of negotiations with source providers, so the extracted information is locally materialized in order to answer user questions. The process model consists of extraction activities (C_1, C_2 and C_3), an activity for reconciling data from two extractors (C_4), an activity for joining critic and film data (C_6), and two activities for performing aggregations and calculating statistics (C_5 and C_7). Users expect data freshness of at least a week.

Although all of them integrate data provided by several cinema sites, their characteristics (e.g. nature of data, system implementation) are very different. Furthermore, freshness factors and metrics as well as user freshness requirements are also different. Consequently, the way data freshness is estimated should be adapted to each application scenario. □

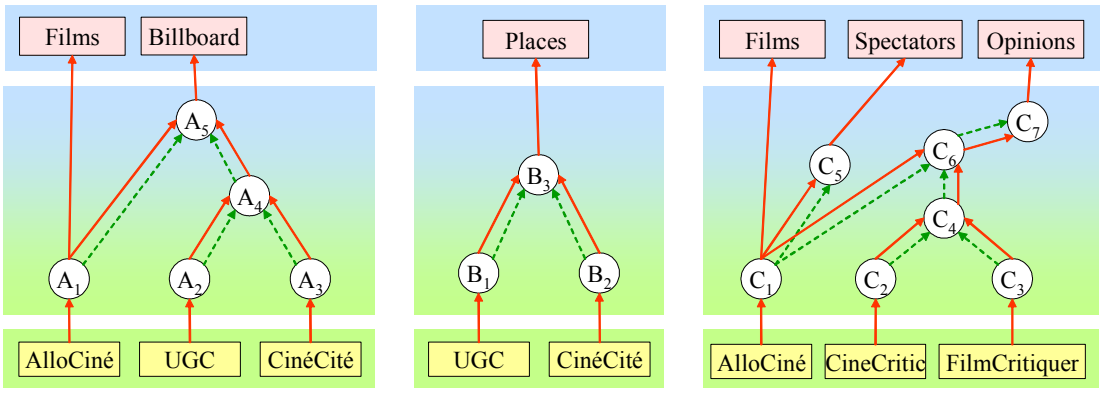

Figure 3.5 – Quality graphs representing the example DISs (omitting property labels)

Next sub-sections describe and apply each instantiation step. We explain the whole method, characterizing the scenarios, determining the properties of each scenario and implementing the overloaded functions. We illustrate the instantiation of the framework to the concrete scenarios introduced in previous example.

3.3. Modeling of scenarios

In this step, DISs processes are classified according to the freshness factor and the four dimensions that influence data freshness. We remind those dimensions (see Sub-section 2.3 of Chapter 2 for details):

- *Nature of data.* This dimension classifies source data according to its change frequency in three categories: *stable*, *long-term-changing* and *frequently-changing* data. When working with frequently changing data, it is interesting to measure how long data can remain unchanged and minimize the delivery of expired data (i.e. evaluate currency). However, when working with data that does not change very often, it is more interesting to measure how old is the data (i.e. evaluate timeliness).
- *Architectural techniques.* This dimension classifies DISs architectural techniques in three categories: *virtual*, *caching* and *materialization* techniques. DISs architectural techniques are very relevant in freshness evaluation because they may introduce significant delays. Specifically, when caching or materializing data, the refreshment frequency causes important delays that should be considered.
- *Synchronization policies.* This dimension classifies DISs synchronization policies according to the interaction between the sources, the DIS and the users (combinations of pull and push policies, in synchronous and asynchronous modes) in 6 categories*: *pull-pull*, *pull/pull*, *pull/push*, *push/push*, *push/pull* and *push-push* policies. Asynchronous modes, with data materialization, may introduce important delays.

This classification is useful in the measurement of processing costs, inter-process delays and source data actual freshness because it summarizes the dimensions that may impact in these properties. A first remark is that the magnitude of source data actual freshness, processing costs and inter-process delays should not be considered in the absolute but compared to freshness expectations. For example, if users may tolerate data freshness of "some days", the processing costs of activities ("some minutes") are negligible; however, if users require data "extremely fresh", the processing costs of activities could be relevant. In order to decide which properties are relevant and which are negligible for a given DIS, the classification according to the taxonomy brings a first idea of magnitudes, which will be taken into account in the calculation of property values. For example, the architectural techniques dimension gives a first idea of the magnitude of inter-process delays.

A scenario is characterized by the freshness factor users are interested in and its classification according to the dimensions of the taxonomy. So, a scenario is described giving four components: (i) freshness factor, (ii) nature of data, (iii) architectural techniques, and (iv) synchronization policies.

> **Example 3.6.** Let's classify the DISs of Example 3.5 according to the freshness factor and the three dimensions of the taxonomy and model the respective scenarios.
>
> Users of DIS_2 want to obtain the same data that is stored at the sources, no matter when the information was updated (when was sold the last ticket), so they are interested in *currency*. However, users of DIS_1 and DIS_3 are mainly interested in seeing information about "recent" films, so they are interested in *timeliness*.
>
> The nature of data is different in the various sources. Cinema descriptive information (UGC and CinéCité sites, accessed by DIS_1) is quite stable, film information (AlloCiné site, accessed by DIS_1 and DIS_3) has a relatively long-term change frequency and place availability (UGC and CinéCité sites, accessed by DIS_2) and critic information (CineCritic and FilmCritiquer sites, accessed by DIS_3) frequently changes.
>
> Concerning DIS implementation, DIS_1 is an interactive process, with virtual techniques and pull synchronous policies; activities are simple JSP queries. DIS_2 is an interactive process, with virtual techniques, that synchronizes the extraction of data from two data sources (pull synchronous with timeout policies); activities are simple copies of data. DIS_3 materializes the data produced by some activities (C_1, C_5, C_6 and C_7), so user queries are answered from materialized data. Data is refreshed periodically: film data is refreshed weekly and critic data daily. Activities are complex cleaning, reconciliation and aggregation processes.
>
> The scenarios for the three previous DISs can be characterized as follows:

* Each configuration is named with the user-DIS policy followed by the DIS-source policy. Asynchronism is represented by a slash (/), synchronism by a dash (-)

- Scenario Sc_1 for DIS_1:
  - Freshness factor: timeliness
  - Nature of data: stable and long-term-changing data
  - Architectural techniques: virtual techniques
  - Synchronization policies: synchronous pull policies (pull-pull)
- Scenario Sc_2 for DIS_2:
  - Freshness factor: currency
  - Nature of data: frequently-changing data
  - Architectural techniques: virtual techniques
  - Synchronization policies: synchronous pull policies (with timeout) (pull-pull)
- Scenario Sc_3 for DIS_3:
  - Freshness factor: timeliness
  - Nature of data: long-term-changing and frequently-changing
  - Architectural techniques: materialization techniques
  - Synchronization policies: asynchronous (periodic) pull policies (pull/pull) □

Next step analyzes the relevant properties of each scenario.

3.4. Identification of appropriate properties

Data freshness is evaluated based on the source data actual freshness, processing cost and inter-process delay properties, but the way of calculating these properties depends on the particular scenario considered. The calculation may be based on DIS properties (e.g. wrapper extraction frequencies, synchronization policies), which may be set as labels of the quality graph or may be measured by used-defined functions. Analogously, the way of combining several freshness values (when an activity has several incoming edges) depends on the scenario and may be based on DIS properties (e.g. data volatility, source reputation). In this sub-section we deal with the identification of the DIS properties involved in such calculations and we discuss their estimation.

3.4.1. Estimation of property values

The calculation of source data actual freshness, processing cost and inter-process delay properties depends on the application scenario. Specifically, we discuss two aspects: (i) the DIS properties that influence their calculation and (ii) the estimation type. The combination of these two aspects should lead to a calculation method.

The first aspect is how to calculate the source data actual freshness, processing costs and inter-process delay properties. Depending on the scenario, different DIS properties may influence their calculation. For example, several delays may compose processing costs of extraction activities in certain DISs [Hull+1996]: the *communication delay* between a source and the activity, the *source query processing delay* and the *activity query processing delay*. Similarly, many delays may influence the inter-process delay, for example when combining data from two sources, activities may hold data from a source while waiting for data from another source. When materializing data, the time passed from last materialization may be an important delay, bounded by the refreshment frequency [Theodoratos+1999]. Properties associated to control flow may be also taken into account, for example, if two activities are synchronized in a way that the latter executes one hour after the former, such synchronization delay should be taken into account. Scheduling methods like Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) [Hiller+1991] can be used for setting delays among activities. When control flow is not driven by data flow, the inter-process delay between two activities might be calculated using properties associated to other activities, as illustrated in the following example.

Example 3.7. Consider the portion of a quality graph shown in Figure 3.6. Activity A_5 is synchronized to execute one unit of time after activity A_3 but it takes as input the data materialized by activity A_2. Activity A_3 is a control routine that does not produce any data. The inter-process delay associated to data edge (A_2,A_5) is calculated as the sum of synchronization delays of control edges (A_2,A_3) and (A_3,A_5) and the processing cost of A_3, i.e. 2 (0+1+1) units of time. □

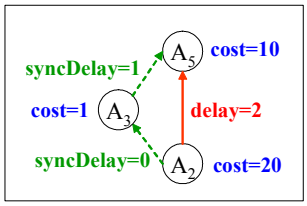

Figure 3.6 – Calculating inter-process delay from control flow properties

Various works propose different estimations for the source data actual freshness (timeliness*), for example: using data timestamps [Braumandl 2003] or the *update frequency* of a source [Naumann+1999]. In sources with access constraints (as restrictions on the query frequency) or dissemination frequencies, such properties should be taken into account. In specific scenarios, the values of source data actual freshness, processing cost or inter-process delay may be directly provided by expert users, system administrators or source providers (set as labels of the quality graph) without need of considering additional DIS properties.

The second aspect concerns the type of estimation, which could be a precise measure of data freshness at this moment or an estimation of data freshness in the average or worst case. The former implies storing metadata in each activity execution (e.g. the extraction time) and executing the evaluation algorithm each time the DIS conveys data (e.g. for each user query). The latter implies keeping statistics and bounds for property values which may be used to calculate upper bounds or average case values of data freshness (without need of calculating freshness when conveying data).

In summary, we should compare the magnitude of source data actual freshness, processing costs and inter-process delays with respect to freshness expectations (aided by the scenario modeled in previous step). For the relevant ones, we should identify the DIS properties that influence their calculation and the desired estimation type in order to obtain their calculation strategies. For example, if the processing cost of extraction activities is relevant, communication delay with sources is the preponderant cost and we want a worst case estimation, a good estimation strategy for the processing cost may consist in keeping statistics of communication costs and taking the maximum. We should study how to acquire each DIS property value. Some properties values may be directly provided by expert users, source providers or DIS administrators, or may be measured or estimated using specialized routines, for example, reading statistics.

> **Example 3.8.** Let's analyze the relevant properties for the three DISs of Example 3.5. Users expect freshness values of "a week" for DIS_1 and DIS_3. With such freshness requirements, the "day" is a good unit for measuring freshness and properties values. Properties with values ranging in "some minutes" or less can be neglected. However, as users expect freshness values of at most "five minutes" for DIS_2, the measurement unit for it should be the "minute" and property values ranging in "some seconds" are relevant.
>
> We want to estimate property values in the worst case. For DIS_1 the processing cost of simple JSP queries (seconds) and the inter-process delay for merging source data (seconds) can be neglected. For DIS_3 the relevant processing costs are due to the reconciliation processes (activities C_4 and C_6), which may require human interaction (to solve conflicts or errors) and may last some days. DIS_3 materializes data, so there is an asynchrony between the data extraction and the data delivery. Sources are queried at different times (because of negotiations with source providers), provoking important inter-process delays between the executions of some activities; then, the refreshment frequencies should be considered. For DIS_2, synchronizing extractors introduces a relevant delay, as the data extracted from a source may be held while waiting for data from the other source. The synchronization timeout is an upper bound for such delay that can be used to estimate it in the worst case. The communication delay with the sources can be important too, as well as the processing cost (seconds) of the merge activity.
>
> For DIS_1 and DIS_3, where users are interested in data timeliness, source data actual freshness (days, weeks or months) is relevant. It can be estimated, in the worst case, as the source update frequency. □

The relevant DIS properties are summarized in Sub-section 3.4.3, as well as the properties necessaries for implementing the combineActualFreshness function. The latter is studied in next sub-section.

* Remember that source data actual freshness is always zero when measuring data currency.

3.4.2. Combination of input values

As argued in Sub-section 3.1, when an activity has several predecessors, the freshness of data coming from them is combined, synthesizing an input freshness value for the activity. That is, given the input values iv_1, iv_2,… iv_n, where iv_i is the freshness value (plus the inter-process delay) of data coming from the i^{th} predecessor, the combineActualFreshness function returns an estimation of the freshness of the whole input:

$cv = CombineActualFreshness\ (iv_1, iv_2, \ldots iv_n)$

The combination strategy depends on the activity semantics. Firstly, if the activity chooses among inputs (e.g. a mediator that returns only the data extracted from the most reputable source), the synthesized value is the input value of such source. Conversely, if the activity merges data from several inputs, the synthesized value should be calculated as a function of the input ones.

A simple combination function considers the worst case, i.e. it returns the maximum of input values. Analogously, the function can return an average (or weighted average) of input values. Typical weights are data volume, data volatility or source reputation). Furthermore, voting strategies can be taken into account (e.g. given more weight to data that is present in more sources). Specialized functions can be defined considering the characteristics of specific scenarios.

Analogously to the calculation of source data actual freshness, processing costs and inter-process delays, the combination function may be based on some DIS properties. We should identify the DIS properties that influence its calculation and determine their calculation strategies.

> **Example 3.9.** Let's analyze the *combineActualFreshness* functions for the three DISs of Example 3.5. For scenarios Sc_2 and Sc_3 the appropriate implementation is taking the maximum of predecessors freshness. However, in scenario Sc_1 users expect to know how fresh is film information, independently to when the cinema data was last updated. We take into account data volatility (movie information is more volatile than actors' information); the strategy consists in ignoring input values of sources providing more stable data. The nature of data dimension is used to define this strategy: expert users assign volatility values. When input activities have the same data volatility, the function returns the maximum input value. □

Next sub-section summarizes the relevant properties and their calculation strategies.

3.4.3. Summary of relevant properties and their calculation

As a result of this step, we produce a list of DIS properties that are necessary for overloading the *getSourceActualFreshness*, *getInterProcessDelay*, *getProcessingCost* and *combineActualFreshness* functions and we sketch the calculation strategies for their implementation. Both properties and strategies will be used in next step for implementing these functions.

Table 3.1 summarizes the DIS properties that are relevant in the calculation of the overloaded functions for the three scenarios of previous examples, indicating, to which graph components they correspond (for example, indicating that only the processing cost of certain activity is relevant) and how to acquire the property values. Table 3.2 summarizes the calculation strategies for them.

		DIS property	Scope	Acquisition
Sc_1	**Source actual freshness**	Update frequency	All sources	Source providers
Sc_2	**Processing cost**	Communication delay	Wrappers	Statistics of connections to sources
		Processing cost	Activity B_3	Statistics of executions
	Inter-process delay	Synchronization timeout	Control edges incoming activity B_3	System administrator
	Combine actual freshness	Data volatility	Data edges	Expert users (at source level); replicated to edges
Sc_3	**Source actual freshness**	Update frequency	All sources	Source providers
	Processing cost	Interaction cost	Activities C_4 and C_6	Statistics of executions
	Inter-process delay	Refreshment frequency	All activities	System administrator

Table 3.1 – DIS properties used for overloading functions

	Scenario Sc_1	Scenario Sc_2	Scenario Sc_3
Processing cost	Neglect	For wrappers (B_1 and B_2): - communication delay with sources + processing cost For merge activity (B_3): - processing cost (worst cases from statistics)	Processing cost (worst case from statistics)
Inter-process delay	Neglect	For edges incoming B_3: - synchronization timeout For other edges: - neglect	For edges outgoing activities: - 1 / refreshment frequency of input node For other edges: - Neglect
Source data actual freshness	1 / update frequency	Neglect	1 / update frequency
Combine actual freshness	Maximum of input values	When different data volatility: - input value of the most volatile input When equal data volatility: - maximum of input values	Maximum of input values

Table 3.2 – Calculation strategies for overloading functions

In next sub-section we describe how to take into account these properties and strategies in the implementation of freshness evaluation algorithms.

3.5. Instantiation of the evaluation algorithm

The freshness evaluation algorithm can be instantiated to adapt it to a specific scenario, overloading the *getSourceActualFreshness*, *getInterProcessDelay*, *getProcessingCost* and *combineActualFreshness* functions, in order to consider in each function, the DIS properties that are most relevant for the scenario (as discussed in previous sub-section).

Generally, the implementation of the overloaded functions is very simple and consists in obtaining the values of some DIS properties or invoking routines to read statistics. However, for some particular scenario, more complex functions may be implemented. As an example, Table 3.3 shows a pseudocode of the *getSourceActualFreshness* function for scenario Sc_1. The function reads the *update frequency* property, acquired from source providers (as specified in Table 3.1) and registered as a label of source nodes of the quality graph. The calculation follows the strategy described in Table 3.2. As another example, the *getProcessingCost* function for scenario Sc_3 should invoke an external function for reading statistics of processing costs of activities (stored for example in a log) and obtaining the maximum. The DIS should store such statistics during execution (no necessarily at each execution).

```
FUNCTION getSourceActualFreshness (G: QualityGraph, A: Node) RETURNS INTEGER
   INTEGER frequency = G.getPropertyValue(A,"UpdateFrequency");
   RETURN 1 / frequency;
END
```

Table 3.3 – Overloading of a function for scenario Sc_1

The framework does not constraint the type of code that can be used in the implementation of overloaded functions; all user-defined functions can be invoked. In addition, the functions implemented for a scenario may be reused for other scenarios (e.g. the function of Table 3.3 may be also used in scenario Sc_3). Furthermore, alternative implementations might be provided in order to support different freshness estimations, for example, if some users need a metric but other users want to analyze another one. This gives additional flexibility to our approach.

A real application scenario is studied in Chapter 5 (Sub-section 3.2). We identify the relevant DIS properties and overload the corresponding functions to instantiate the freshness evaluation algorithm. .

Next sub-section discusses the propagation of freshness expected values which allows to constraint source providers on the freshness of source data.

3.6. Propagation of freshness expectations

Analogously to the propagation of freshness actual values, we can propagate freshness expected values from target to source nodes. The propagated freshness expected values may help the DIS designer to know the freshness that he should ask the source providers for. A direct application of this propagation strategy is the comparison of alternative data sources in order to select the one that provides the freshest data.

Sub-section 3.1 presented the propagation of freshness actual values from source to target nodes. The propagation of freshness expected values is quite similar but presents some additional problems. In this sub-section we discuss the propagation of freshness expected values and we present the corresponding propagation algorithm.

The propagation algorithm calculates the following property:

– *Expected freshness*: It is an estimation of the expected freshness for data outgoing a node (*Efreshness* for short).

The propagation principle is to traverse the quality graph (in sense inverse to the edges) calculating the expected freshness for data outgoing each node, i.e. the maximum freshness value that may be tolerated for the data produced by the node, in order to achieve freshness expectations. Intuitively, while for calculating actual values we add processing costs and inter-process delays to source data actual freshness, for calculating expected values, we should subtract them from target data expected freshness. A first intuitive propagation algorithm starts with target data expected freshness, and for each node, subtract the processing cost of the node and the inter-process delay with the predecessor node. Next example illustrates the idea:

> **Example 3.10.** Consider the quality graph of Figure 3.7a as input for the propagation of freshness expected values. Actual freshness was propagated with the basic algorithm described in Sub-section 3.1, taking the maximum as combination function (as illustrated in Example 3.4). The expected freshness for (A_3, T_1) is calculated as the target data expected freshness of T_1, i.e. 30 units of time. The expected freshness for (A_1,A_3) is calculated as the expected freshness of (A_3, T_1) minus the cost of A_3 minus the inter-process delay of (A_1,A_3), obtaining a value of 21 (30-4-5) units of time. The other values are calculated analogously. Figure 3.7b shows the output quality graph. □

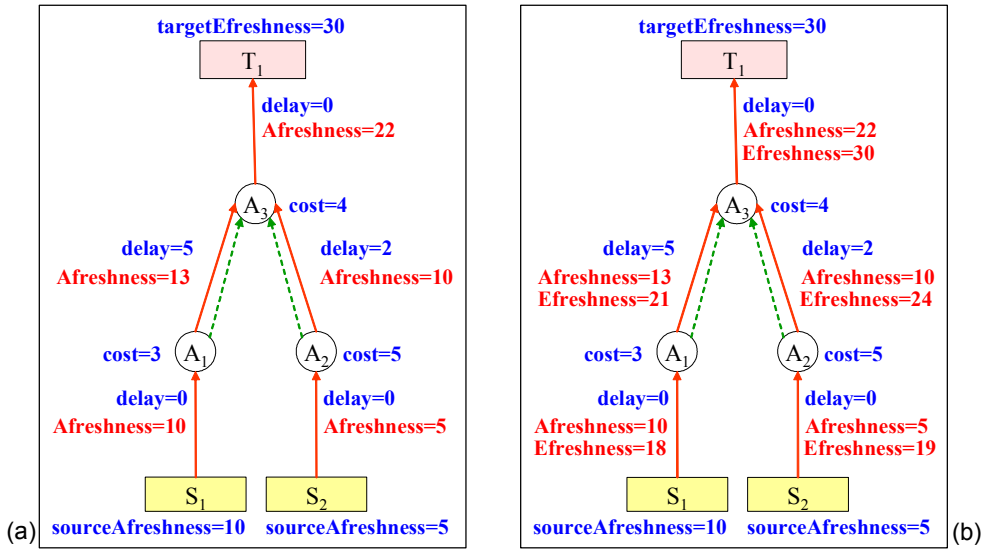

Figure 3.7 – Propagation of freshness expected values: (a) input quality graph, and (b) output quality graph

In previous example, freshness expected values are propagated in the same way to all predecessors. However, in certain scenarios, different values should be propagated to each predecessor. Such differentiation depends on the

semantics of the combination function (combineAcutalFreshness), which synthesizes an input freshness value for each node. The following example illustrates this idea:

> **Example 3.11.** Consider again the quality graph of Figure 3.7a but suppose that freshness actual values were propagated using a different combination function which returns the freshness of input data coming from S_1 (for example, because of data volatility). Note that it does not matter which is the freshness of data coming from S_2, if freshness of data from S_1 is good enough, freshness expectations are achieved. For example, if the source data actual freshness of S_2 is 5000 units of time instead of 5 units of time, the actual freshness of (A_3, T_1) continues being 22 units of time, which does not surpass freshness expectations. So, the propagation of freshness expected values in this quality graph should respect this intuition, i.e. the expected freshness for (A_2,A_3) should be infinite. □

When a node has several predecessors, its expected freshness is decomposed among the predecessors obtaining an expected value for each predecessor. We define the *decomposeExpectedFreshness* function for performing such decomposition. Given an input value *iv* corresponding to the expected freshness for data outgoing a node (minus the processing cost of the node), the decomposeExpectedFreshness function returns a set of values $\{v_1, v_2, \ldots v_n\}$ where v_i is the expected freshness value for data incoming the node from the i^{th} predecessor:

$$\{v_1, v_2, \ldots v_n\} = \text{decomposeExpectedFreshness (iv)}$$

Previous example motivated that the decomposition of expected values should be coherent to the combination of actual values. Specifically, the decomposeExpectedFreshness function should have the inverse effect of the combineActualFreshness function. The idea is that, if the combination function is applied to the propagated expected values, the synthesized value should satisfy freshness expectations. To see this, let v be the freshness expected value for a node and $v_1, v_2, \ldots v_n$ the freshness expected values propagated to the node predecessors, i.e. decomposeExpectedFreshness (v) = $\{v_1, v_2, \ldots v_n\}$; then, the combination function applied to $v_1, v_2, \ldots v_n$ should return a smaller value than v. In other words, the decomposition function should warranty that combineActualFreshness ($v_1, v_2, \ldots v_n$) $\leq v$.

The expected values propagated to predecessors (i.e. $v_1, v_2, \ldots v_n$) should be the greatest values that allows achieving freshness expectations (i.e. v). In other words, the combination function applied to smaller values should return a synthesized value that is smaller than freshness expectations. For that reason, we require the combineActualFreshness and decomposeExpectedFreshness functions to be monotonic.

The decomposition function is the solution to the following optimization problem:

$$\text{Maximize:} \quad (v_1, v_2, \ldots v_n)$$

$$\text{subject to:} \quad \text{combineActualFreshness } (v_1, v_2, \ldots v_n) \leq v$$

Unfortunately, for some combination functions the decomposition function cannot be easily deduced. Furthermore, in some situations, we cannot completely determine expected values and we only obtain equations with some degree of freedom, which are not as useful as those illustrated in previous examples. Next example illustratres one of such cases.

> **Example 3.12.** Consider that the combination function for the quality graph of Figure 3.7a performs an average of both input values, i.e. for activity A_3, combineActualFreshness (18,12) = 15. The freshness expected values for predecessors of activity A_3 should be the greater values that verify that combineActualFreshness (v_1,v_2) $\leq$ 30-4, i.e. (v_1+v_2)/2 $\leq$ 26.
>
> In this case, values v_1 and v_2 are not determined and there is a space of solutions that verifies the condition. Specifically, solutions are of the form (v, 52-v}. □

In cases where the optimal decomposition function cannot be exactly determined, the function should be implemented for returning some values that could be useful for a particular scenario but knowing that it will be more restrictive than necessary, for example, decomposeExpectedFreshness (v) = {v,v…v}.

Once the decomposition function has been identified, for a specific scenario, the propagation of expected values is analogous to that of actual values. It can be sketched as follows (see Figure 3.8): First, for each successor activity ($C_1 \ldots C_m$), we obtain the expected freshness. We take the most restrictive value (the minimum) and we subtract the processing cost. Then, we decompose such value in a set of values, one for each predecessor activity ($P_1 \ldots P_n$), and we subtract, from each value, the inter-process delay with the corresponding predecessor.

1. u_1 = ExpectedFreshness(A,C_1) …
   u_m = ExpectedFreshness(A,C_m)
2. v = min {u_1,…u_m} – ProcessingCost(A)
3. {v_1,…v_n} = DecomposeExpectedFreshness (v)
4. ExpectedFreshness (P_1,A) = v_1 - InterProcessDelay (P_1,A) …
   ExpectedFreshness (P_n,A) = v_n - InterProcessDelay (P_n,A)

Figure 3.8 – Expected freshness propagation strategy

The previous strategy is implemented in an algorithm for propagating freshness expectations: *ExpectedFreshnessPropagation* (see Algorithm 3.2). The algorithm first spans target nodes, obtaining target data expected freshness and storing the expected freshness for each incoming edge. Then, the algorithm traverses activity nodes, calculating the minimum expected freshness of outgoing edges and subtracting the processing cost. The resulting value is decomposed obtaining an expected value for each predecessor, in a list (valList). Each value of the list is subtracted of the corresponding inter-process delay and stored for the incoming edge. A pseudocode of the algorithm can be sketched as shown in Algorithm 3.2.

The *decomposeExpectedFreshness* function is an abstract function that should be overloaded to implement the adequate decomposition strategy.

```
FUNCTION ExpectedFreshnessPropagation (G: QualityGraph) RETURNS QualityGraph
   INTEGER efreshness, value, m;
   FOR EACH target node T DO
      efreshness = getTargetExpectedFreshness(G,T);
      EDGE e = data edge incoming T in G
      G.addProperty(e,"ExpectedFreshness",efreshness);
   ENDFOR;
   FOR EACH activity node A in inverse topological order of G DO
      m = min ({G.getPropertyValue(e,"ExpectedFreshness") / e is a data edge outgoing A in G})
      value= m – getProcessingCost(G,A);
      HASHTABLE valList = edges incoming A in G;
      decomposeExpectedFreshness (value, valList);
      FOR EACH data edge e incoming A in G DO
         efreshness = valList.get(e) – getInterProcessDelay(G,e);
         G.addProperty(e,"ExpectedFreshness",efreshness);
      ENDFOR;
   ENDFOR;
   RETURN G;
END
```

Algorithm 3.2 – Basic algorithm for propagating freshness expected values

Next sub-section presents some direct applications of both types of freshness propagations.

3.7. Usages of the approach

The algorithms for propagating actual and expected values of data freshness proposed in previous sub-sections can be used in different contexts, either in order to evaluate data freshness in an existing DIS or to determine constraints for DIS development. This sub-section summarizes the freshness evaluation approach by briefly presenting some usages of the approach for different purposes and at different development stages. Some of these usages are developed in the remaining of the chapter (Sections 4 and 5) and in Chapter 5.

3.7.1. Evaluating data freshness at different phases of the DIS lifecycle

The data freshness evaluation approach can be used at different phases of the DIS lifecycle, e.g. at design, production or maintenance phases. Depending on the phase, a quality graph may represent an existing DIS (implemented, operative) or a specification of a DIS (not yet implemented). In the former, precise property values (source data actual freshness, processing costs and inter-process delays) can be obtained during DIS execution. In the latter, property values should be estimated, generally based on cost models. In both cases, property values can be upper bounds (e.g. worst case processing costs) so the evaluation algorithm will obtain upper bounds for data freshness.

Some usages of the approach, in an existing DIS are:

- At query evaluation: We can evaluate data freshness during query evaluation in order to obtain precise freshness values of the conveyed data (possibly labeling data with its freshness values). The actual freshness evaluation algorithm can be integrated to the query evaluation process. Precise property values can be obtained during DIS execution. In that case, we obtain the actual freshness of the conveyed data.
- At query planning: We can evaluate data freshness before executing a query in order to predict the freshness of data that may be returned in response to the query. Data freshness evaluation should be based on estimations of property values (e.g. using cost models or statistics of previous query evaluations). This is the case of virtual DIS (e.g. mediation systems) where the calculation operations (activities) are decided for each user query. Query optimizers use costs models and statistics for predicting operations costs.
- At DIS monitoring: We can evaluate data freshness offline (after executing several queries) in order to monitor if the DIS has quality problems. Data freshness evaluation should be based on estimations of property values (generally based on statistics of DIS executions). The data freshness evaluation algorithm can be integrated to a quality auditing tool.

Some usages of the approach, in a DIS specification are:

- At DIS design: We can evaluate data freshness in a (conceptual or logical) model of the DIS in order to validate the model. Data freshness evaluation should be based on estimations of DIS properties (e.g. using cost models).
- At DIS maintenance: When doing modifications to DIS design (e.g. changing implementation of some activities or adding new components for providing additional processing), we can evaluate data freshness of the new model in order to validate the changes or decide if they are convenient. Statistics or cost models for the new/modified components can be used as estimations of DIS properties.
- At DIS reengineering: When reengineering a DIS (possibly because of quality problems), we can evaluate data freshness in a new version of the DIS (e.g. substituting some activities by more performing ones) in order to validate the modifications. Data freshness evaluation should be based on statistics of DIS executions and estimations of property values for the new components.

Note that for query evaluation and DIS monitoring data freshness evaluation is performed *a posteriori*, i.e. after executing queries, however, for query planning data freshness evaluation is performed *a priori*, i.e. before executing queries. In a DIS specification, data freshness evaluation is also performed *a priori*.

3.7.2. Different applications of the evaluation approach

Both types of propagations (of actual and expected values), either a priori or a posteriori, can be used for different purposes, ranging from simple quality assessment (e.g. for informing users of actual freshness) to quality improvement (e.g. for analyzing improvement actions). In this sub-section we discuss several usages of the approach.

Data freshness assessment

A first use of the approach consists simply on evaluating data freshness and communicating it to the correspondent actors (designers, administrators, users, source providers). The evaluation results can be used for different purposes, for example:

- *Communicating data freshness to users*: Propagating freshness actual values, from sources to targets, we obtain metadata (data freshness) that qualifies conveyed data. Freshness values can be communicated to

users either as labels of conveyed data (in response to a query) or as upper bounds (before executing the query). In both cases, providing data freshness values to users represent a value-added to the data.

- *Estimating data freshness*: Using estimations of property values (e.g. based on cost models or statistics) we can estimate the freshness of data that may be returned by a DIS. Results can validate design decisions or motivate the development of more performing components. So, data freshness may have impact in early design steps. In addition, at maintenance phase, freshness estimations may serve for monitoring the DIS.
- *Specifying constraints for source data actual freshness*: Propagating freshness expected values, from targets to sources, we obtain freshness expected values for source data. These values may serve as constraints (upper bounds for source data actual freshness) for warranting the achievement of freshness expectations. They can be communicated to source providers when negotiating the service. In that way, freshness expectations also may have impact in early design steps.
- *Specifying constraints for DIS development*: The comparison among target data expected freshness and source data actual freshness serves to determine constraints for DIS implementation. Their difference, if positive, indicates the longest period of time that DIS processes are allowed to spend in manipulating data, i.e. it is an upper bound for the execution delay of the DIS which includes the processing costs of activities and the inter-process delays among them. Examples of constraints are: maximal processing costs for activities, minimal execution frequencies for activities, minimal refreshment frequencies for materialized data and minimal access frequencies for data sources.

Comparison of different DIS implementations

A direct application of the freshness evaluation approach is the comparison among different quality graphs. The application consists in evaluating data quality in each quality graph and selecting the one that produces data with the highest quality (data freshness can be the unique quality criteria or can be balanced with other ones). The quality graphs may differentiate, for example, in the access to alternative data sources, in the use of different activities and in the interaction among activities (data and control flows).

- *Selecting a quality graph:* Freshness actual values can be propagated, from sources to targets, obtaining measures for comparing among graphs. The comparison can be done at different moments, for example, at design phase, we can compare different (conceptual or logical) models for the DIS, at production phase (query planning), we can compare alternative execution plans, at maintenance phase we can compare different modifications to DIS implementation.
- *Selecting a data source:* Given a model for the DIS, accessing to some generic sources (specification of the data types that must be provided by sources), we can propagate freshness expected values to sources obtaining constrains for source data. Then, candidate sources can be compared in order to select the one that provides data with the highest quality. Analogously, the comparison can be done at the different phases of the DIS lifecycle.

In this line, we used the freshness evaluation approach in a mediation application in the context of data personalization. Complementing a procedure that automatically generates mediation queries for accessing alternative data sources, the evaluation approach was used for estimating data freshness of the generated queries in order to select the best one for a given user [Kostadinov+2004]. This application is described in Chapter 5 (Sub-section 3.1).

Data freshness analysis

The data freshness propagation algorithms can be used as auditing tools for measuring the quality of the data produced by the DIS and analyzing the DIS based on data quality. Analysis includes checking if freshness expectations are satisfied and, if not, determining the critical points of the DIS. Both types of propagations are useful:

- *Checking if freshness expectations are satisfied for targets*: We can compare freshness values obtained with the *ActualFreshnessPropagation* algorithm with those expected by users (target data expected freshness). The comparison allows finding the data targets for which freshness expectations are satisfied and those for which freshness expectations are overdrawn. This result may be important for informing users about the expectations that are not satisfied (inviting them to relax expectations).

- *Checking if freshness expectations are satisfied by sources*: Analogously, we can compare freshness values obtained with the *ExpectedFreshnessPropagation* algorithm with those provided by sources (source data actual freshness). The comparison allows finding the data sources for which freshness expectations are satisfied and those for which freshness expectations are overdrawn. This result may be important for informing source providers about data that is not satisfactory for users (inviting them to provide a better service. i.e. fresher data).

In both cases, we identify portions of the DIS (targets, sources, paths from sources to targets) that may be analyzed in detail and possibly improved. Analysis can be done at design phase in order to validate the model, at production phase in order to monitor the DIS and at maintenance phase in order to suggest modifications. Sub-sections 4.1 to 4.3 discuss data freshness analysis.

Data freshness improvement

If freshness expectations are overdrawn, different improvement actions can be taken in order to enforce data freshness. Certain strategies intent to improve DIS implementation (e.g. substituting an activity by a more performing component, synchronizing activities for reducing delays, powering hardware for accessing more frequently or performingly to data sources) and other ones intent to obtain fresher source data (e.g. substituting a data source, negotiating with source data providers for relaxing access constraints). Sub-section 4.4 discusses improvement actions.

Next section discusses data freshness enforcement. It utilizes both types of quality propagations for analyzing the quality graph, building a diagnostic on the achievement of freshness expectations and highlighting the bottlenecks for data freshness achievement, i.e. the portions of the quality graph that should be improved in order to enforce data freshness. We also present some improvement actions that can be applied to those bottlenecks in order to enforce data freshness.

4. Data freshness enforcement

The DIS should provide, for each target node, the data freshness expected by the users. The freshness evaluation approach, presented in previous section, allows checking if a DIS satisfies freshness expectations. To do such validation, we can calculate the freshness actual values for target nodes and compare them with those expected by users. If freshness actual values are lower than expected values then freshness can be guaranteed. If freshness actual values are greater than expected values, freshness expectations are not achieved and some improvement actions should be followed in order to enforce freshness. Examples of improvement actions are improving the implementation of activities in order to reduce their processing cost and synchronizing activities in order to reduce inter-process delays among them. The identification of critical paths (portions of the quality graph that represent bottlenecks for freshness calculation) allows concentrating improvement actions in the subsets of activities that cause freshness expectations to be overdrawn.

In this section we present an approach for analyzing the quality graph at different levels of abstraction in order to identify critical paths and apply improvement actions to the nodes in the path.

4.1. Top-down analysis of data freshness

In this sub-section we propose an approach for analyzing data freshness in a top-down way, inspired from OLAP browsing mechanisms. Generally, OLAP users start analyzing aggregated data and when abnormal or warming situations are suspected, they perform drill-downs on identified data in order to obtain further information for understanding the phenomena, determining causes and conceiving solutions. Analogously, the proposed approach first analyzes data freshness in a high-level quality graph (macroscopic representation of the DIS) and analyzes more detailed quality graphs (microscopic representations of the DIS) when further details are necessaries for enforcing data freshness. Data freshness analysis consists in propagating data freshness values as discussed in previous section (ActualFreshnessPropagation algorithm), checking if freshness expectations can be achieved. If freshness values satisfy freshness expectations the DIS is appropriate for users freshness needs, however, if freshness values overdraw freshness expectations, detailed information about DIS activities (i.e. a lower level quality graph) may be useful for identifying bottlenecks and proposing improvement actions.

We consider a hierarchy of representations of the DIS processes (quality graphs at different abstraction levels) and two operators: *level-up* and *level-down*, which allow changing from a representation to another one, i.e. changing the level of abstraction. Figure 3.9 shows a hierarchy of quality graphs composed of three levels; the highest-level graph has a unique activity representing the whole DIS, allowing the visualization of the data sources that participate in the calculation of data targets; the intermediate-level graph abstracts macro activities and the lowest-level graph shows details about activity logics (i.e. detailed data and control flows).

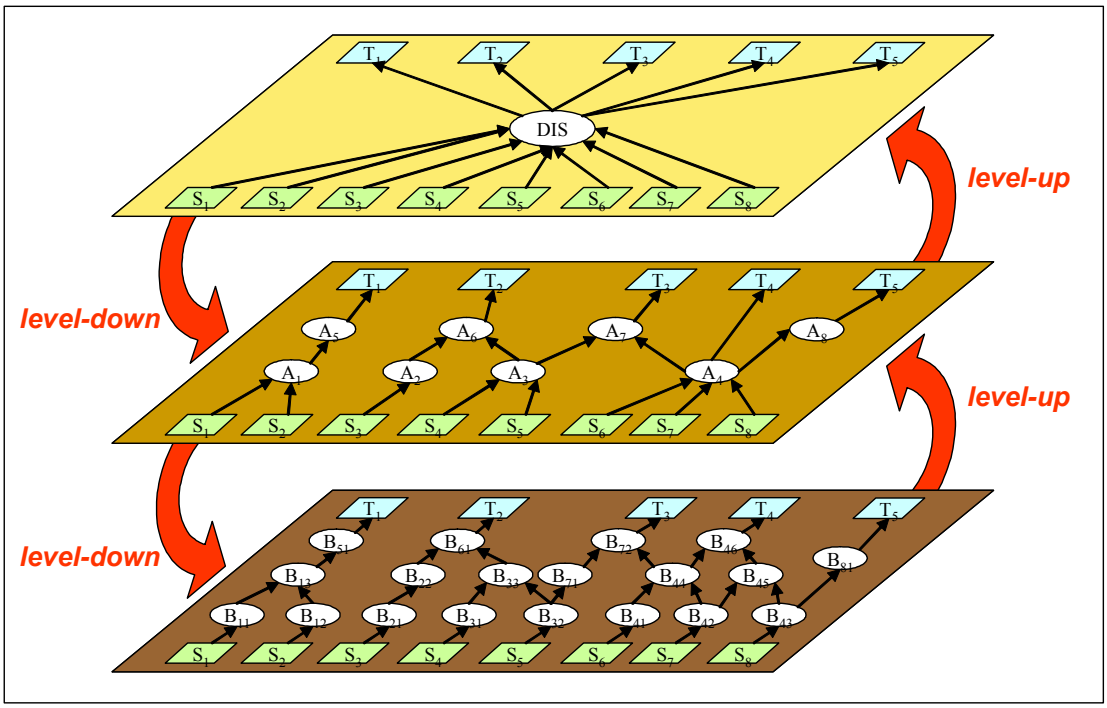

Figure 3.9 – Hierarchy of quality graphs

Next sub-sections model the hierarchy of quality graphs, describe the operations for browsing inside the model and aggregating processing costs and inter-process delays for high-level graphs, and discuss data freshness evaluation at different levels of the hierarchy.

4.1.1. Hierarchy of quality graphs

For representing the hierarchy of quality graphs we need two components: (i) an ordered sequence of quality graphs (each graph increasing the level of abstraction of its predecessor in the sequence), and (ii) a relation among activities of consecutive levels, indicating that they correspond to the same conceptual task.

For representing the latter, we consider a hierarchy of activities where the root represents the whole DIS and each level represents DIS activities at a given level of abstraction. High-level activities abstract high-level tasks while lower-level activities show the processing details of the tasks. Ascending in the hierarchy correspond to abstracting task behaviors while descending in the hierarchy can be interpreted as decomposing an activity in more detailed and precise sub-tasks. For example, certain extraction activity may be decomposed in a sequence of sub-activities that perform the connection to the source, the execution of an extraction query, the filtering of irrelevant or erroneous data and the storage of the resulting data. Figure 3.10 shows the hierarchy of activities for the DISs of Figure 3.9. Note that all branches of the hierarchy tree have the same length.

We represent the hierarchy of quality graphs with the above-mentioned components: a hierarchy of activities and an array of quality graphs:

> **Definition 3.7 (hierarchy of quality graphs).** A *hierarchy of quality graphs* is a pair (AH, GH) where *AH* is a tree of activities representing the hierarchy of activities and *GH* is a list of quality graphs representing the levels of abstraction. □

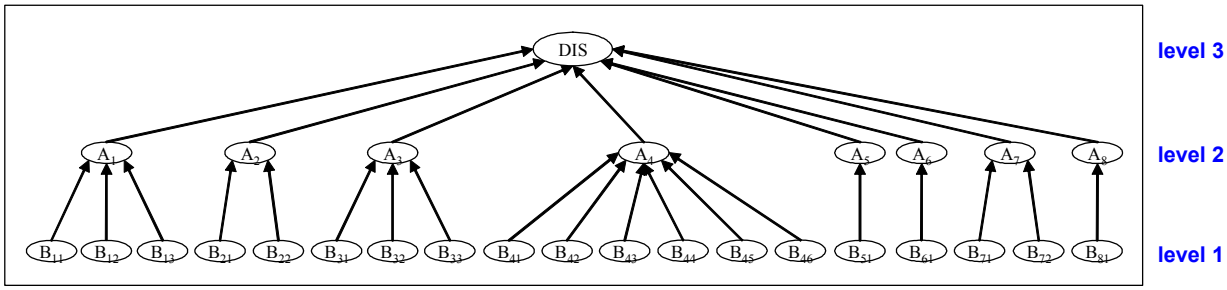

Figure 3.10 – Hierarchy of activities

When referring to two consecutive graphs in the hierarchy, the lowest-level graph is called *detailed graph* and the highest-level graph is called *summarized graph*.

In the following, we show how to build a summarized graph from a detailed one, how to aggregate property values and how to evaluate data freshness in it. To this end, we need to make some hypotheses about the hierarchy of quality graphs and the propagation of data freshness. The first hypothesis concerns the semantics of activities. It states that the execution of sub-activities must have the same effect than executing their parent activity, i.e. take the same inputs and produce the same outputs (both in data and control flows). Consequently, there may exist an edge between two activities (A and B) if and only if there is an edge between a sub-activity of A and a sub-activity of B. The second hypothesis states that data and control flow of the lowest-level quality graph must warranty that there will be no cycles at any abstraction level. In other works, if there is a path among a sub-activity descendent of A and a sub-activity descendent of B (being A and B two activities of the same high-level quality graph), it must not exist a path among any sub-activity descendent of B and any sub-activity descendent of A. The third hypothesis states that sibling sub-activities must be connected in the corresponding quality graph. This hypothesis is necessary to calculate processing costs of high-level activities, as will be explained in next sub-section. The last hypothesis states that the combineActualFreshness function returns the maximum of input values and consequently, the decomposeExpectedFreshness function returns the same value to all predecessors. This hypothesis is necessary to prove that freshness values can be propagated at all abstraction levels. Using different combination and decomposition functions we should make analogous proofs that those that will be shown in Sub-section 4.3. We have chosen these functions because they are the most typically used in several types of DIS, for example [Naumann+1999] [Braumandl 2003].

Given a quality graph of level L and a hierarchy of activities, the quality graph of level L+1 can be built using the *buildLevelGraph* function (see Algorithm 3.3). The method starts adding source and target nodes (which are the same for all levels) and then adds the activity nodes of level L+1. An edge between two nodes is created whether there is an edge between children activities. Given the quality graph of lowest-level and the hierarchy of activities, all higher-level quality graphs can be built invoking successively the *buildLevelGraph* function.

The GraphHierarchy class has methods for managing the array of quality graphs and the tree of activities:

```
FUNCTION getLevel (G: QualityGraph) RETURNS INTEGER
FUNCTION getLevel (A: Activity) RETURNS INTEGER
FUNCTION getLevelGraph (level: INTEGER) RETURNS QualityGraph
FUNCTION getLevelGraph (A: Activity) RETURNS QualityGraph
FUNCTION levelUp (G: QualityGraph) RETURNS QualityGraph
FUNCTION levelDown (G: QualityGraph) RETURNS QualityGraph
FUNCTION getSubActivities (A: Activity) RETURNS ActivitySet
FUNCTION getParentActivity (A: Activity) RETURNS Activity
FUNCTION getSiblingActivities (A: Activity) RETURNS ActivitySet
```

The *getLevel* functions return the level (in the hierarchy) of a quality graph, giving either the graph or one of its activities. The *getLevelGraph* functions return the quality graph of a level, giving either the level or one of its activities. The *levelUp* and *levelDown* functions return the quality graphs that are immediately upper (less detail) and lower (more detail) in the hierarchy than a given quality graph, respectively. The getSubActivities, getParentActivity and getSiblingActivities functions return the corresponding neighbors in the hierarchy of activities.

```
FUNCTION buildLevelGraph (AH: TREE OF Activity, int level, G: QualityGraph)
                                RETURNS QualityGraph
   QualityGraph Q;
   Q.insertNodes (G.getSourceNodes());
   Q.insertNodes (G.getTargetNodes());
   Q.insertNodes (AH.getLevelNodes(level));
   FOR EACH edge e=(B1,B2) in G DO
      IF (B1 is a source or target node) THEN A1=B1;
      ELSE A1 = AH.getParentActivity(B1);
      IF (B2 is a source or target node) THEN A2=B2;
      ELSE A2 = AH.getParentActivity(B2);
      Q.insertEdge (A1,A2,e.getType());
   ENDFOR;
   RETURN Q;
END
```

Algorithm 3.3 – Method for building the hierarchy of quality graphs

In order to evaluate data freshness in high-level quality graphs, processing costs and inter-process delays should be calculated from the processing costs and inter-process delays of lower-level graphs (which is analogous, following the analogy with OLAP applications, to the aggregation of measures, i.e. the roll-up operation). Next sub-section discusses their calculation.

4.1.2. Labeling of high level quality graphs

In this sub-section we deal with the calculation of processing costs and inter-process delays of high-level quality graphs. In order to simplify the analysis, we label quality graphs with such properties*. The labeling of the lowest-level quality graph can be done executing the getSourceActualFreshness, getProcessingCost and getInterProcessDelay functions discussed in Section 3.

Intuitively, the processing cost of an activity in the summarized graph should be the time necessary to execute all its sub-activities in the detailed graph. In other words, the processing cost should equal the extent of time between the start of execution of the initial sub-activity (the first one in starting execution) and the end of execution of the final sub-activity (the last one in finishing execution). In order to find initial and final sub-activities, and consequently calculating the processing cost, sub-activities are scheduled, respecting the processing costs and inter-process delays of the detailed graph, as follows:

> **Definition 3.8 (execution schedule)**. An *execution schedule* is a function that maps sub-activities to time intervals of the form $<t_A,T_A>$, with $0 \leq t_A \leq T_A$; t_A is called the *starting time* and T_A is called the *ending time* of the execution of sub-activity A. A schedule verifies the following conditions: (i) for each sub-activity, the length of its interval equals its processing cost, (ii) for each pair of consecutive sub-activities, the separation between their intervals equals the inter-process delay between them, (iii) for some sub-activity, $t_A=0$. The sub-activity (or sub-activities) that satisfies the last condition is called *initial sub-activity*. The sub-activity (or sub-activities) that has maximum ending time is called *final sub-activity*. □

Execution schedules can be visualized using Gantt charts as illustrated in Figure 3.11b. Time-intervals are represented by rectangles, which lengths indicate processing costs. Separations among rectangles represent inter-process delays.

* The same analysis can be done without labeling quality graphs but overloading the getSourceActualFreshness, getProcessingCost and getInterProcessDelay functions for high-level quality graphs. For simplifying the understanding of the approach, we calculate property values, label quality graphs with them and overload functions for simply reading the property values.

Execution schedules can be computed using the Critical Path Method (CPM) [Hiller+1991]. The method uses a *potential task graph*, which is a directed labeled graph whose nodes represent tasks (activities) and whose edges represent precedence constraints, labeled with time constraints (delays). The potential task graph for a set of sub-activities $\{B_1,\ldots B_n\}$ can be built from the detailed graph. The problem of finding starting times for sub-activities reduces to finding most expensive paths in the potential task graph. The Ford algorithm [Hiller+1991] computes starting times with order $O(m^3)$ being m the number of edges of the graph. Ending times are calculated adding processing costs to starting times.

The following example illustrates the use of the execution schedule for calculating processing costs.

Example 3.13. Consider an activity B with 3 sub-activities: B_1, B_2 and B_3. Figure 3.11a shows a portion of the quality graph that contains the sub-activities. Figure 3.11b shows the execution schedule corresponding to the quality graph. The starting times of B_1, B_2 and B_3 are 1, 0 and 4 respectively, while their ending times are 3, 2 and 8 respectively. The initial sub-activity is B_1 and the final sub-activity is B_3. □

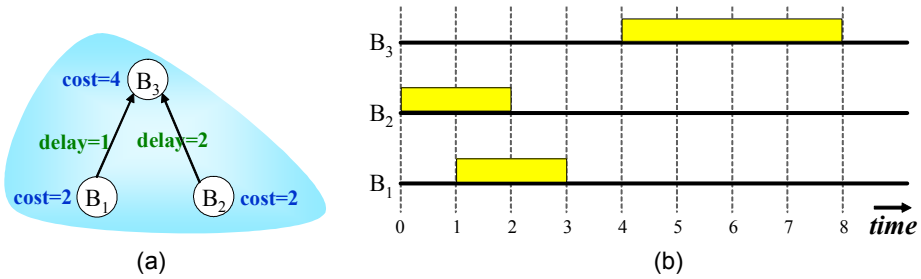

Figure 3.11 – Calculation of starting and ending times of sub-activities: (a) portion of a quality graph, (b) execution schedule

Now, we can formalize the calculation of processing costs in a high-level graph. Let A be an activity of a summarized graph S and let $\{A_1\ldots A_m\}$ be the sub-activities of A in a detailed graph D. The processing cost of A is calculated as the time passed from the starting time of its initial sub-activity (zero) to the ending time of its final sub-activity. Figure 3.12 shows the calculation formula.

$$\begin{aligned} \text{ProcessingCost}(A) &= \text{EndingTime}(\text{FinalSubActivity}(A)) \\ &= \max\{\text{EndingTime}(A_1),\ldots \text{EndingTime}(A_m)\} \end{aligned}$$

Figure 3.12 – Calculation of processing costs in high-level graphs

In order to calculate inter-process delays, we should consider delays among sub-activities but also take into account the starting and ending times of sub-activities. Intuitively, the inter-process delay between two activities A and B in a summarized graph should be the difference of time between the executions of their sub-activities in the detailed graph. Specifically, inter-process delay should equal the extent of time between the ending time of the final sub-activity of A and the starting time of the initial sub-activity of B. To this end, the execution schedules of activities A and B can be concatenated, respecting the inter-process delay among sub-activities, as shown in Figure 3.13b.

Example 3.14. Consider two activities A and B of a summarized graph with sub-activities $\{A_1,A_2\}$ and $\{B_1,B_2\}$ respectively. Figure 3.13a shows a portion of the detailed graph that contains the sub-activities; shadow zones highlight siblings in the hierarchy of activities. Figure 3.13b shows the execution schedules (global time is added at the top of the figure for facilitating the visualization of inter-process delays). As the inter-process delay of (B_1,A_2) is 8 units of time, the schedules are shifted in 8 units of time respect to global time. The inter-process delay of (B,A) is easily seen in the graphic: it is 3 units of time. □

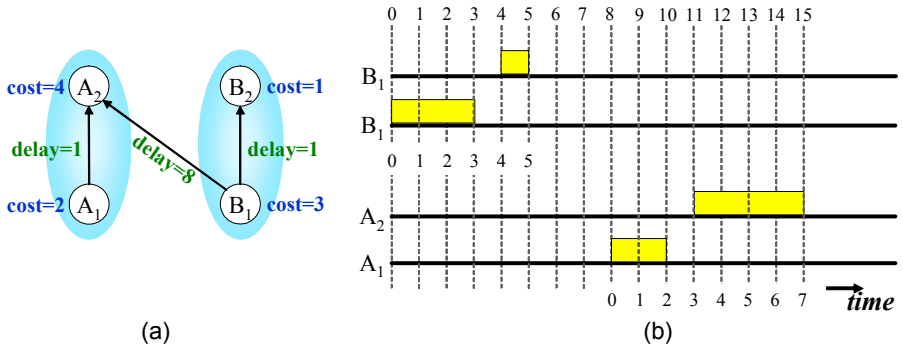

Figure 3.13 – Concatenation of execution schedules: (a) portion of a quality graph, (b) execution schedules

Now, we can formalize the calculation of inter-process delays in a high-level graph. Let A and B be two activities of a summarized graph S and let $\{A_1 \ldots A_m\}$ and $\{B_1 \ldots B_n\}$ be their sub-activities in a detailed graph D, respectively. The inter-process delay between two sub-activities A_i and B_j may be greater or equal to the inter-process delay between A and B. Equality is achieved when A_i is a final sub-activity of A and B_i is an initial sub-activity of B. In the rest of the cases, the difference of time between the ending time of a final sub-activity of A and the ending time of A_i, and the difference of time between the starting time of B_j and the starting time of an initial sub-activity of B (which is time 0), are both subtracted from the inter-process delay of (A_i,B_j). Note that inter-process delays can be negative in a summarized graph if B starts execution before A finishes. Figure 3.14 shows the calculation of inter-process delays.

$$\begin{aligned}\text{InterProcessDelay}(A,B) = \max \{ & \text{InterProcessDelay}(A_i,B_j) - \\ & (\text{EndingTime}(\text{FinalSubActivity}(A)) - \text{EndingTime}(A_i)) - \text{StartingTime}(B_j) \\ & / (A_i,B_j) \text{ is a data edge of D, } 1 \le i \le m, 1 \le j \le n \ \}\end{aligned}$$

Figure 3.14 – Calculation of inter-process delays in high level graphs

The maximum is taken in the case there exist several data edges among sub-activities of A and sub-activities of B, despite the same value should be obtained if inter-process delays are coherently assigned in the detailed graph.

Next example summarizes the calculation of processing costs and inter-process delays.

> **Example 3.15.** Figure 3.15 shows a summarized graph S with three activities (A_3, A_6 and A_7) and a detailed graph D with six sub-activities (B_{31}, B_{32}, B_{33}, B_{61}, B_{71} and B_{72}); both graphs are simplified versions or those of Figure 3.9. Figure 3.15a shows processing costs and inter-process delays of D while Figure 3.11b shows starting and ending times of sub-activities (e.g. the starting time of B_{33} is 4 units of time and its ending time is 8 units of time). The processing cost of A_3 is calculated as the ending time of its final sub-activity (B_{33}), i.e. 8 units of time. The inter-process delay between A_3 and A_7 is calculated as the inter-process delay between B_{32} and B_{71} minus the difference of ending times among B_{33} and B_{32} minus the starting time of B_{71}, obtaining a value of -3 (3 – (8-2) – 0) units of time. The other processing costs and inter-process delays are calculated analogously, following the formulas of Figure 3.12 and Figure 3.14. □

Next sub-section discusses the use of such labels for calculating data freshness in high-level quality graphs.

4.1.3. Data freshness evaluation at different abstraction levels

Having built the hierarchy of quality graphs and labeled the graphs with processing cost, inter-process delay and source data actual freshness properties, we can evaluate data freshness at the different abstraction levels using the ActualFreshnessPropagation algorithm. Lemma 3.1 proves that freshness values calculated in a summarized graph are greater or equal to those calculated in a detailed graph. In other words, the propagation of freshness values in high-level graphs brings upper bounds for data freshness.

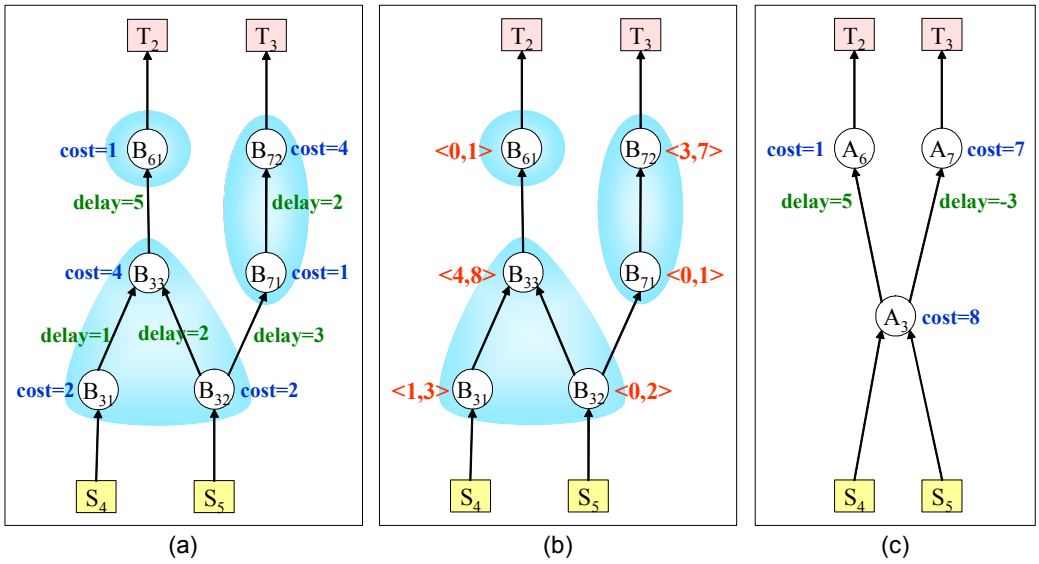

Figure 3.15 – Calculation of processing costs and inter-process delays in high level graphs: (a) detailed graph, (b) starting and ending times in the detailed graph, and (c) summarized graph

Lemma 3.1. Given a summarized graph S obtained applying the level-up operator over a detailed graph D, it is verified that:

∀ target node V (being B the predecessor of V in D and being A the parent activity of B in S) . Freshness(A,V) ≥ Freshness(B,V)

The proof is deferred to Sub-section 4.3 because we need to use some results presented in such sub-section.

In most situations, the level-up method preserves freshness actual values propagated to target nodes, i.e. the same freshness values are obtained if evaluation is performed in a summarized or in a detailed graph. Exceptions may occur when an activity has several initial and final sub-activities, as illustrated in next example. In such exceptions, some freshness values of the summarized graph may be greater than those of the detailed graph.

Example 3.16. Consider the propagation of freshness actual values in the summarized and detailed graphs of Figure 3.16 (S and D respectively). Note that actual freshness of data incoming V_2 is preserved, but actual freshness of data incoming V_1 is considerably lower in the detailed graph. The reason is that there is a path from S_2 to T_1 in S but there is no path between them in D. □

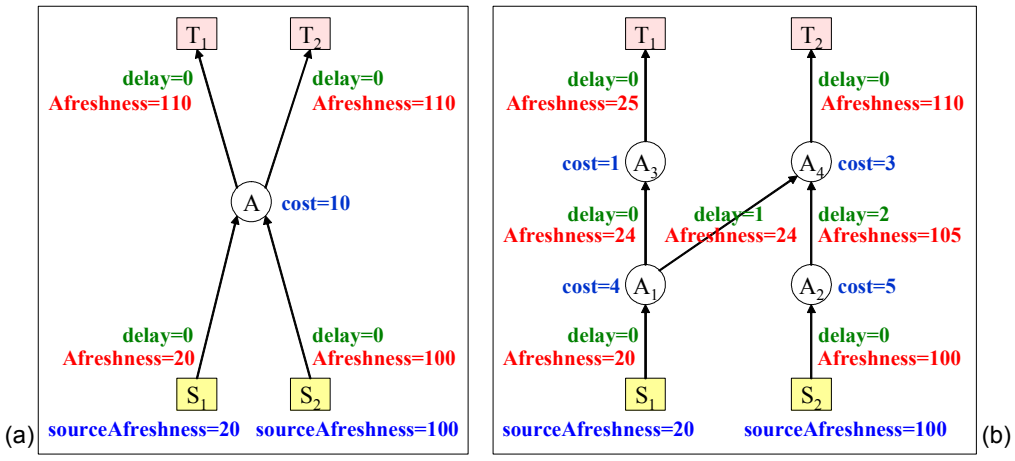

Figure 3.16 – Propagating data freshness: (a) summarized graph, (b) detailed graph

Lemma 3.1 warranties that data freshness analysis realized in high-level graphs are valid in lower-level ones, i.e. if freshness expectations are satisfied in high-level graphs they are achieved in lower-level ones. When expectations are overdrawn, the level-down method allows decomposing activities and analyzing the graph in detail in order to obtain more accurate freshness values. Furthermore, better adapted improvement actions can be applied to detailed graphs in order to enforce data freshness (which will be discussed in Sub-section 4.4).

The *levelUp* and *levelDown* functions allow browsing in the hierarchy of quality graphs, but they return the whole graph of the corresponding level. In order to analyze in detail certain activities (e.g. for trying to optimize them and then enforce data freshness) we need methods for browsing portions of the quality graphs. Next sub-section presents such methods.

4.2. Browsing among quality graphs

There are two basic operations to refine the analysis of an activity: focus+ and zoom+. Focusing on an activity means concentrating the analysis in the activity ignoring the other ones, in other words, keeping only the portion of the quality graph that is relevant for the analysis: the activity and its neighbors. Zooming in an activity means analyzing the sub-activities that compose the given activity, in other words, descending a level in the hierarchy of quality graphs and showing the sub-activities and their neighbors. The focus– and zoom–methods achieve the inverse effects.

> **Example 3.17.** Consider the top-down analysis of Figure 3.17 over the hierarchy of quality graphs of Figure 3.9. We start the analysis at the high-level quality graph (first panel). The second panel results of applying the zoom+ operation to the node representing the DIS; it shows the main DIS activities. Suppose that we want to analyze activity A_4, for example, for trying to reduce its processing cost. To this end, we apply the focus+ operation to A_4 (third panel), showing A_4 and its neighbor nodes. The last panel results of applying the zoom+ operation to A_4; it shows the sub-activities that compose it and their neighbor nodes. Note that after applying some operations, source and target nodes might represent activities. Applying zoom–and focus– operations in reverse order we obtain the highest-level quality graph. □

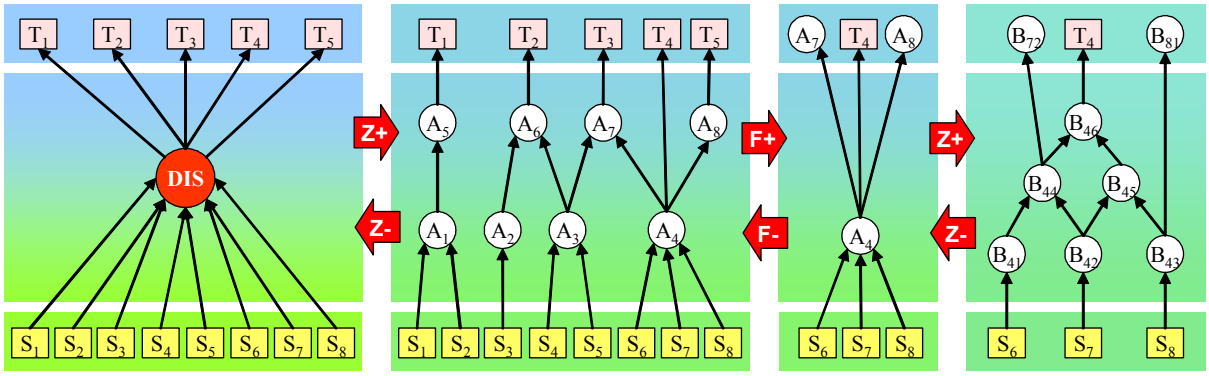

Figure 3.17 – Operations for browsing the hierarchy of quality graphs: zoom+ (Z+), zoom– (Z–), focus+ (F+) and focus– (F–)

Algorithm 3.4 shows the pseudocodes of the focusing methods. The *focus+* method returns a sub-graph of the given quality graph containing the given activity, its predecessors and successors and the edges among them, i.e. it returns the sub-graph induced by an activity and its neighbors. The *neighborSubGraph* method (see Algorithm 3.6) returns the portion of the quality graph of a given level induced by a given set of nodes and their neighbors. The *focus–* method returns a quality graph (of the same abstraction level than the given one) containing the activity, its sibling nodes (in the hierarchy of activities), their predecessors and successors and the edges among them, i.e. it returns the sub-graph of the level graph induced by an activity, its siblings and their neighbors.

Algorithm 3.5 shows the pseudocodes of the zooming methods. The *zoom+* method returns a quality graph (of the immediately inferior abstraction level) containing the sub-activities of the given activity, their predecessors and successors and the edges among them, i.e. it returns the sub-graph of the inferior-level graph induced by sub-activities and their neighbors. The *zoom–* method returns a quality graph (of the immediately superior abstraction level) containing the parent activity of the given one, their predecessors and successors and the edges among them, i.e. it returns the sub-graph of the upper-level graph induced by the parent activity and their neighbors.

```
FUNCTION focus+ (G: QualityGraph, A: Activity) RETURNS QualityGraph
    RETURN neighborSubGraph (G,{A});
END
FUNCTION focus– (GH: GraphHierarchy, A: Activity) RETURNS QualityGraph
    QualityGraph G = GH.getLevelGraph(A);
    ActivitySet SA = GH.getSiblingActivities(A)) ∪ {A};
    RETURN neighborSubGraph (G,SA);
END
```

Algorithm 3.4 – Focusing methods

```
FUNCTION zoom+ (GH: GraphHierarchy, Activity A) RETURNS QualityGraph
    QualityGraph G1 = GH.getLevelGraph(A);
    QualityGraph G2 = GH.levelDown(G1);
    ActivitySet SA = GH.getSubActivities(A);
    RETURN neighborSubGraph (G2,SA);
END
FUNCTION zoom– (GH: GraphHierarchy, Activity B) RETURNS QualityGraph
    QualityGraph G1 = GH.getLevelGraph(A);
    QualityGraph G2 = GH.levelUp(G1);
    Activity A = GH.getParentActivity(B);
    RETURN neighborSubGraph (G2,{A});
END
```

Algorithm 3.5 – Zooming methods

```
FUNCTION neighborSubGraph (G: QualityGraph, SA: ActivitySet) RETURNS QualityGraph
    QualityGraph Q;
    Q.insertNodes (SA);
    FOR EACH node A of SA DO
        Q.insertNodes (G.getPredecessors(A));
        Q.insertNodes (G.getSuccessors(A));
    ENDFOR;
    FOR EACH edge e incoming or outgoing nodes of SA in G DO
        Q.insertEdge (e);
    ENDFOR;
    RETURN Q;
END
```

Algorithm 3.6 – A method for building sub-graphs induced by a set of nodes and their neighbors

The focus+ and zoom+ methods allow refining the analysis of an activity. In order to refine the analysis of a (connected) sub-graph of the quality graph, analogous methods can be defined for focusing and zooming in a set of connected activities. Their signatures are the following:

```
FUNCTION focus+ (G: QualityGraph, SA: ActivitySet) RETURNS QualityGraph
FUNCTION focus– (GH: GraphHierarchy, SA: ActivitySet) RETURNS QualityGraph
FUNCTION zoom+ (GH: GraphHierarchy, SA: ActivitySet) RETURNS QualityGraph
FUNCTION zoom– (QH: GraphHierarchy, SA: ActivitySet) RETURNS QualityGraph
```

Pseudocodes are analogous.

When expectations are overdrawn, the zoom+ method allows decomposing an activity in order to study it in detail and eventually finding accurate improvement actions (which will be discussed in Sub-section 4.4), for example, substituting a sub-activity for a more performing component. Furthermore, if ending times are decreased in the detailed graph, the processing cost of the activity may be decreased in the summarized graph too. Then, the zoom+ method represents an interesting opportunity for enforcing data freshness. In addition, the focus+ method allows focusing on one activity and ignoring the others, which is more manageably than enormous graphs with details about all activities.

Next sub-section presents a method for finding the activities that constitute bottlenecks for data freshness calculation. The zoom+ and focus+ methods may be applied to these activities, targeting the analysis of data freshness.

4.3. Determination of critical paths

For each target node, it may exist a path (along the data flow), starting at a source node, for which the freshness of delivered data can be obtained adding all the inter-process delays and processing costs of the nodes in the path, to the source data actual freshness of the source node. This path is called the *critical path* of the target node and represents the bottleneck for data freshness. The following example presents the intuition of critical paths.

> **Example 3.18.** Consider the quality graph of Figure 3.18. The freshness of data produced by activity A_6 (delivered to target T_2) can be calculated adding the source data actual freshness of source S_1 (0), plus inter-process delays (0,0,10,20) and processing costs (30,60,30,5) in the path from S_1 passing by activities A_1,A_3,A_5 and A_6, i.e. 0 + (0,0,10,20) + (30,60,30,5) = 155. So, this path is a critical path for T_2. □

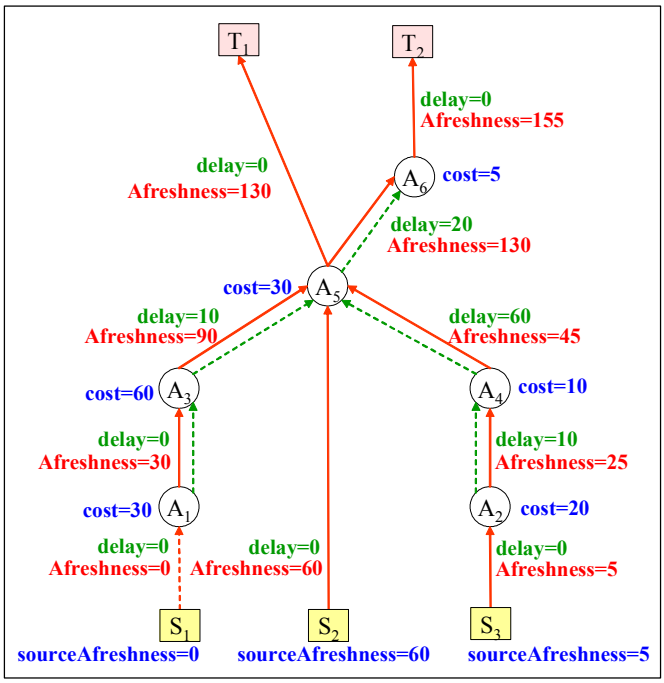

Figure 3.18 – An example of critical paths

As previously argued, the freshness of the data delivered to the user may be improved optimizing the design of the activities in order to reduce their processing costs or synchronizing the activities in order to reduce the inter-process delays among them. Sometimes the changes can be concentrated in the critical path, i.e. reducing source data actual freshness, processing costs and inter-process delays of the nodes of the critical path. This motivates the analysis and determination of critical paths.

The existence of critical paths depends on the definition of the combination function, i.e. for certain functions the critical path may not exist. In this sub-section, we consider that the combination function returns the maximum

of input values and we prove that for such function the critical path always exists. Similar analysis can be done for finding bottlenecks for other combination functions.

Before defining critical paths we define the concepts of data path and path freshness.

Definition 3.9 (data path). 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 3.10 (path freshness). 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 actual value propagated along the path (ignoring other nodes of the graph), i.e. it is the sum of source data actual freshness of the source node, the processing costs of the nodes in the path and the inter-process delays among the nodes*:

$$\text{PathFreshness}([A_0,\ldots A_p]) = \text{SourceActualFreshness}(A_0) + \sum_{x=0..p} \text{ProcesssingCost}(A_x) + \sum_{x=1..p} \text{InterProcessDelay}(A_{x-1},A_x)$$ □

The critical path for an activity is the data path (from a source node to the activity) that determines the freshness of the activity, i.e. the freshness actual value for the activity is equal to the path freshness. In other words, if we ignore other nodes and we calculate data freshness only using the critical path we obtain the same freshness value. We define a critical path as follows:

Definition 3.11 (critical path). 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, e.g. C) equals the path freshness.

$$\text{Freshness}(A_p,C) = \text{PathFreshness}([A_0,\ldots A_p])$$

Given a target node T_i, a *critical path* for T_i is the critical path of its predecessor activity. □

The following lemma states the existence of at least a critical path for each activity node.

Lemma 3.2. Given an activity node A_p, with successor A_{p+1}, there exists a data path $[A_0,A_1,\ldots A_p]$ from a source node A_0 to A_p that verifies:

$$\text{Freshness}(A_p,A_{p+1}) = \text{PathFreshness}([A_0,\ldots A_p])$$

Proof:

According to the ActualFreshnessPropagation algorithm, freshness of data produced by node A_p (delivered to A_{p+1}) is obtained adding the processing cost of A_p to the combination of freshness values (plus inter-process delays) of predecessors ($P_1\ldots P_n$). The considered combination function returns the maximum input values.

$$\text{Freshness}(A_p,A_{p+1}) = \text{ProcessingCost}(A_p) + \max(\{\text{Freshness}(P_1,A_p) + \text{InterProcessDelay}(P_1,A_p), \ldots \text{Freshness}(P_n,A_p) + \text{InterProcessDelay}(P_n,A_p)\})$$

Let A_{p-1} be the predecessor that achieves the maximum in the previous formula. Then:

$$\text{Freshness}(A_p,A_{p+1}) = \text{ProcessingCost}(A_p) + \text{InterProcessDelay}(A_{p-1},A_p) + \text{Freshness}(A_{p-1},A_p)$$

Applying the same reasoning to A_{p-1}, to its predecessor achieving the maximum (A_{p-2}) and so on, we obtain:

$$\begin{aligned}\text{Freshness}(A_p,A_{p+1}) = {} & \text{ProcessingCost}(A_p) + \text{InterProcessDelay}(A_{p-1},A_p) \\ & + \text{ProcessingCost}(A_{p-1}) + \text{InterProcessDelay}(A_{p-2},A_{p-1}) + \ldots \\ & + \text{ProcessingCost}(A_1) + \text{InterProcessDelay}(A_0,A_1) \\ & + \text{SourceActualFreshness}(A_0)\end{aligned}$$

By definition of path freshness (Definition 3.10) we have:

$$\text{Freshness}(A_p,A_{p+1}) = \text{PathFreshness}([A_0,A_1,\ldots A_p])$$ ■

* Remember that, as defined in Sub-section 3.1, processing costs are associated to all nodes (with zero value for source and target nodes) and inter-process delays are associated to all data edges (with zero value for edges outgoing source nodes or incoming target nodes).

The following lemma and its corollary provide a way of finding critical paths by computing the path freshness of all data paths from source nodes. They state that critical paths are those that have the greatest path freshness.

Lemma 3.3. Given an activity node A_p, with successor A_{p+1}, all data paths $[A_0,\dots A_p]$ from source nodes verify:

$$\text{Freshness}(A_p,A_{p+1}) \geq \text{PathFreshness}([A_0,\dots A_p])$$

Proof by induction in the length of the path:

→ *Basis step*: for data paths of length 2 $[A_0,A_1]$. $\text{Freshness}(A_1,A_2) \geq \text{PathFreshness}([A_0,A_1])$

Proof:

$$\text{Freshness}(A_1,A_2) = \text{ProcessingCost}(A_1) + \max(\{\text{Freshness}(X,A_1) + \text{InterProcessDelay}(X,A_1) \mid X \text{ is a predecessor of } A_1\})$$

Taking a particular predecessor achieves a smaller or equal value. Then, for A_0:

$$\begin{aligned}\text{Freshness}(A_1,A_2) &\geq \text{ProcessingCost}(A_1) + \text{Freshness}(A_0,A_1) + \text{InterProcessDelay}(A_0,A_1)\\ &= \text{ProcessingCost}(A_1) + \text{SourceActualFreshness}(A_0) + \text{InterProcessDelay}(A_0,A_1)\\ &= \text{PathFreshness}([A_0,A_1]) \quad \blacksquare\end{aligned}$$

→ *Inductive step:* Assume that for all data paths of length h≥2 $[A_0,\dots A_{h-1}]$. $\text{Freshness}(A_{h-1},A_h) \geq \text{PathFreshness}([A_0,\dots A_{h-1}])$ in order to prove that for data paths of length h+1 $[A_0,\dots A_{h-1},A_h]$. $\text{Freshness}(A_h,A_{h+1}) \geq \text{PathFreshness}([A_0,\dots A_{h-1},A_h])$

Proof:

$$\text{Freshness}(A_h,A_{h+1}) = \text{ProcessingCost}(A_h) + \max(\{\text{Freshness}(X,A_h) + \text{InterProcessDelay}(X,A_h) \mid X \text{ is a predecessor of } A_h\})$$

Taking a particular predecessor achieves a smaller or equal value. Then, for A_{h-1}:

$$\text{Freshness}(A_h,A_{h+1}) \geq \text{ProcessingCost}(A_h) + \text{Freshness}(A_{h-1},A_h) + \text{InterProcessDelay}(A_{h-1},A_h)$$

Using inductive hypothesis:

$$\begin{aligned}\text{Freshness}(A_h,A_{h+1}) &\geq \text{ProcessingCost}(A_h) + \text{PathFreshness}([A_{0,\dots}A_{h-1}]) + \text{InterProcessDelay}(A_{h-1},A_h)\\ &= \text{PathFreshness}([A_0,\dots A_{h-1},A_h]) \quad \blacksquare\end{aligned}$$

Corollary:

Given an activity node A_p, with successor A_{p+1}, the freshness of data produced by A_p is equal to the maximum path freshness of the paths from a source node, i.e.:

$$\text{Freshness}(A_p,A_{p+1}) = \max\{\text{PathFreshness}([A_0,\dots A_p]) \mid [A_0,\dots A_p] \text{ is a data path from a source node}\}$$

Proof:

By Lemma 3.3 the freshness of data produced by A_p is greater or equal to the path freshness of all data paths from source nodes and by Lemma 3.2 we know that there exists a path that verifies the equality (a critical path). So such path is the one with greatest path freshness. ■

Corollary of Lemma 3.3 suggest an effective method for calculating critical paths: finding the data paths with greatest path freshness. They can be computed using the Critical Path Method (CPM) [Hiller+1991], labeling a *potential task graph* with processing costs, inter-process delays and source data actual freshness (the potential task graph is similar to the quality graph but all labels are associated to edges). The Bellman algorithm [Hiller+1991] computes critical paths with order O(m) being m the number of edges of the graph.

We can now prove Lemma 3.1 using Lemma 3.2 and Lemma 3.3, as follows:

Proof of Lemma 3.1. The lemma stated that given a summarized graph S obtained applying the level-up operator over a detailed graph D, it is verified that:

$\forall$ target node V (being B the predecessor of V in D and being A the parent activity of B in S) . Freshness(A,V) $\geq$ Freshness(B,V)

Proof:

Let $[X,B_{11},\ldots B_{1m_1},B_{21},\ldots B_{2m_2},\ldots B_{n1},\ldots B_{nm_n}]$ be the critical path built by Lemma 3.2, with $B_{nm_n}=B$, and let A_i be the parent activity of $B_{i1},\ldots B_{im_i}$, $1\leq i\leq n$, with $A_n=A$. Note that the path $[X,A_1,\ldots A_n]$ must exist in S because, by construction (BuildLevelGraph method), an edge $(B_{im_i},B_{(i+1)1})$, $1\leq i<n$, cause the edge (A_i,A_{i+1}) to belong to S.

On the one hand, by Lemma 3.2 and definition of path freshness we have:

$$\begin{aligned}\text{Freshness}(B,V) &= \text{PathFreshness}([X,B_{11},\ldots B_{1m_1},B_{21},\ldots B_{2m_2},\ldots B_{n1},\ldots B_{nm_n}]) \\ &= \text{SourceActualFreshness}(X) + \text{ProcessingCost}(B_{11}) + \text{InterProcessDelay}(B_{11},B_{12}) \\ &\quad + \ldots + \text{ProcessingCost}(B_{nm_n}) \end{aligned} \quad (1)$$

On the other hand, by Lemma 3.3 and definition of path freshness we have:

$$\begin{aligned}\text{Freshness}(A,V) &\geq \text{PathFreshness}([X,A_1,\ldots A_n]) \\ &= \text{SourceActualFreshness}(X) + \text{ProcessingCost}(A_1) + \text{InterProcessDelay}(A_1,A_2) \\ &\quad + \ldots + \text{ProcessingCost}(A_n) \end{aligned} \quad (2)$$

We define α_i=StartingTime(B_{i1}) and β_i=EndingTime(B_{im_i}), $1\leq i\leq n$. Note that:

$$\text{ProcessingCost}(B_{i1}) + \text{InterProcessDelay}(B_{i1},B_{i2}) + \ldots + \text{ProcessingCost}(B_{im_i}) = \beta_i - \alpha_i \quad (3)$$

According to the calculation of processing costs and inter-process delays for summarized graphs (Figure 3.12 and Figure 3.14 respectively) we have:

$$\text{ProcessingCost}(A_i) = \text{EndingTime}(\text{FinalSubActivity}(A_i))$$

$$\begin{aligned}\text{InterProcessDelay}(A_i,A_{i+1}) = \max\ (\{&\text{InterProcessDelay}(B_{ij},B_{(i+1)k}) - \text{EndingTime}(\text{FinalSubActivity}(A_i)) + \\ &\text{EndingTime}(B_{ij}) - \text{StartingTime}(B_{(i+1)k}) \ /\ (B_{ij},B_{(i+1)k}) \text{ is a data edge of D}\})\end{aligned}$$

In particular, the edge $(B_{im_i},B_{(i+1)1})$ achieves a smaller or equal value in previous formula. By algebraic manipulation we have:

$$\begin{aligned}\text{InterProcessDelay}(B_{im_i},B_{(i+1)1}) &\leq \text{EndingTime}(\text{FinalSubActivity}(A_i)) + \text{InterProcessDelay}(A_i,A_{i+1}) - \beta_i + \alpha_{i+1} \\ &= \text{ProcessingCost}(A_i) + \text{InterProcessDelay}(A_i,A_{i+1}) - \beta_i + \alpha_{i+1} \end{aligned} \quad (4)$$

Substituting (3) and (4) in (1) we obtain:

$$\begin{aligned}\text{Freshness}(B,V) &\leq \text{SourceActualFreshness}(X) + (\beta_1 - \alpha_1) \\ &\quad + (\text{ProcessingCost}(A_1) + \text{InterProcessDelay}(A_1,A_2) - \beta_1 + \alpha_2) + (\beta_2 - \alpha_2) + \ldots \\ &\quad + (\text{ProcessingCost}(A_{n-1}) + \text{InterProcessDelay}(A_{n-1},A_n) - \beta_{n-1} + \alpha_n) + (\beta_n - \alpha_n) \\ &= \text{SourceActualFreshness}(X) - \alpha_1 + \text{ProcessingCost}(A_1) + \text{InterProcessDelay}(A_1,A_2) + \ldots \\ &\quad + \text{ProcessingCost}(A_{n-1}) + \text{InterProcessDelay}(A_{n-1},A_n) + \beta_n\end{aligned}$$

As $\alpha_1 \geq 0$ and $\beta_n \leq$ ProcessingCost(A_n):

$$\begin{aligned}\text{Freshness}(B,V) \leq \text{SourceActualFreshness}(X) &+ \text{ProcessingCost}(A_1) + \text{InterProcessDelay}(A_1,A_2) + \ldots \\ &+ \text{ProcessingCost}(A_{n-1}) + \text{InterProcessDelay}(A_{n-1},A_n) + \text{ProcessingCost}(A_n)\end{aligned}$$

Finally, using (2):

$$\text{Freshness}(B,V) \leq \text{PathFreshness}([X,A_1,\ldots A_n]) \leq \text{Freshness}(A,V) \quad \blacksquare$$

As proved in previous lemma, when calculating freshness in a more detailed graph we may obtain more precise values than in a summarized graph. In addition, some improvement actions can be better applied in a more detailed graph which brings more specific information about processing costs and inter-process delays.

Next sub-section discusses some strategies for enforcing data freshness when user freshness expectations cannot be achieved.

4.4. Improvement actions

If freshness actual values are greater than expected values, freshness expectations are not achieved and some improvement actions should be followed in order to enforce freshness. Freshness actual values can be improved optimizing the design and implementation of the activities in order to reduce their processing cost, synchronizing the activities in order to reduce the inter-process delay among them, or negotiating with source providers in order to obtain fresher source data. In addition, freshness expected values can be relaxed negotiating with users. Consequently, improvement actions can be the response to one of the following objectives: (i) reduce processing costs, (ii) reduce synchronization delays, (iii) reduce source data actual freshness, or (iv) augment target data expected freshness.

The result of applying an improvement action will be changes in the topology or the properties of the quality graph (i.e. changes in nodes, edges and labels). Elementary actions are:

- addNode (N: Node) – This action adds a node to the quality graph. The node is not yet connected to other nodes (it has no incoming nor outgoing edges) and has not labels.
- addEdge (e: Edge) – This action adds an edge to the quality graph. The edge has not labels.
- addProperty (N: Node, P: Property, value: Object) – This action associates a label property=value to a node of the quality graph. If the property is already associated to the node, its value is updated. The data type of the value corresponds to the data type of the property.
- addProperty (e: Edge, P: Property, value: Object) – This action associates a label property=value to an edge of the quality graph. If the property is already associated to the node, its value is updated. The data type of the value corresponds to the data type of the property.
- removeNode (N: Node) – This action removes a node (and all incoming and outgoing edges, as well as associated properties) from a quality graph.
- removeEdge (e: Edge) – This action removes an edge (and all associated properties) from a quality graph.
- removeProperty (N: Node, P: Property) – This action deletes a property from a node of the quality graph.
- removeProperty (e: Edge, P: Property) – This action deletes a property from an edge of the quality graph.

All these elementary actions are member methods of the QualityGraph class.

Elementary actions can be combined, conforming macro actions that solve typical improvement strategies. There is a great variety of macro actions that can be defined. Their feasibility depends on the particular application scenario, i.e. an action that considerably improves freshness in a DIS may have no impact on another one. In this sub-section we illustrate some typical macro actions (without trying to be exhaustive) that can be applied in a great variety of DISs. Examples of macro actions are:

- replaceNode (G: QualityGraph, N: Node, N': Node, NP': NodePropertySet) – This action replaces a node (N) by a new one (N'), keeping the same edges. A set of properties (NP') is associated to the new nodes. The action can be used, for example, for replacing an activity by a more performing one (reducing processing cost) or replacing a source by another one (reducing source data actual freshness).
- replaceSubGraph (G: QualityGraph, NS: NodeSet, ES: EdgeSet, NS': NodeSet, ES': EdgeSet, NP': NodePropertySet, EP': EdgePropertySet) – This action replaces a set of nodes and a set of edges conforming a sub-graph (NS and ES) by new set of nodes (NS'), connected by a set of edges (ES'). Sets of properties (NP' and EP') are associated to the new nodes and edges respectively. The action can be used, for example, 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.
- decomposeNode (G: QualityGraph, N: Node, NS': NodeSet, ES': EdgeSet, NP': NodePropertySet, EP': EdgePropertySet) – This action replaces a node (N) by a new set of nodes conforming a sub-graph (NS) which represent refined tasks, connected by a set of edges (ES'). Sets of properties (NP' and EP') are associated to the new nodes and edges respectively. The action can be used, for example, for replacing an activity by a set of activities representing refined tasks (which can be optimized or synchronized separately).

- parallelizeNodes (G: QualityGraph, NS: NodeSet, ES': EdgeSet, EP': EdgePropertySet) – This action replaces a set of edges among a path of nodes (NS), corresponding to a sequential execution, by edges (ES') connecting the nodes according to a parallel execution. A set of properties (EP') is associated to the new edges. The action can be used, for example, for parallelizing the execution of certain activities in order to reduce global processing cost.
- changeNodesProperties (G: QualityGraph, NP': NodePropertySet) – This action changes property values of a set nodes. The argument NP' is a set of 3-uples of the form <node,property,value>. The action can be used, for example, for changing the processing cost property after optimizing the implementation activities, changing the source data actual freshness property after negotiation with source providers or changing DIS policies (e.g. refreshment frequencies) that will impact inter-process delays.
- changeEdgesProperties (e: Edge, EP': EdgePropertySet) – This action changes the property values of a set of edges. The argument EP' is a set of 3-uples of the form <edge,property,value>. The action can be used, for example, for changing the inter-process delay property after synchronizing activities.

In order to facilitate the easy implementation of macro actions, we define two additional macro actions (that can be used as templates) which manage (add and remove respectively) sets of nodes, edges and labels, invoking elementary actions. They have the following signatures:

PROCEDURE addSets (G: QualityGraph, NS: NodeSet, ES: EdgeSet, NP: NodePropertySet, EP: EdgePropertySet)

PROCEDURE removeSets (G: QualityGraph, NS: NodeSet, ES: EdgeSet, NP: NodePropertySet, EP: EdgePropertySet)

Inputs consists of a quality graph, a set of nodes, a set of edges, a set of triplets <node,property,value> and a set of triplets <edge,property,value>. Pseudocodes are shown in Algorithm 3.7.

```
PROCEDURE addSets (G: QualityGraph, NS: NodeSet, ES: EdgeSet, NP: NodePropertySet,
                   EP: EdgePropertySet)
   FOR EACH node N in NS DO
      G.addNode(N);
   FOR EACH node e in ES DO
      G.addEdge(e);
   FOR EACH triplet (node,prop,value) in NP DO
      G.addProperty(node,prop,value);
   FOR EACH triplet (edge,prop,value) in EP DO
      G.addProperty(edge,prop,value);
END
```

```
PROCEDURE removeSets (G: QualityGraph, NS: NodeSet, ES: EdgeSet, NP: NodePropertySet,
                      EP: EdgePropertySet)
   FOR EACH node N in NS DO
      G.removeNode(N);
   FOR EACH edge e in ES DO
      G.removeEdge(e);
   FOR EACH triplet (node,prop,value) in NP DO
      G.removeProperty(node,prop);
   FOR EACH triplet (edge,prop,value) in EP DO
      G.removeProperty(edge,prop);
END
```

Algorithm 3.7 – Template macro actions

Algorithm 3.8 shows the pseudocodes of the replaceNode, replaceSubGraph, decomposeNode, parallelizeNodes, changeNodesProperties and changeEdgesProperties macro actions. They are quite similar: they build the appropriate sets (nodes, edges, labels) and invoke the addSets and removeSets template actions.

```
PROCEDURE replaceNode (G: QualityGraph, N: Node, N': Node, NP': NodePropertySet)
    EdgeSet ES';
    EdgePropertySet EP';
    FOR EACH edge e=(B,N)^T incoming N in G DO
        e' = (B,N')^T;
        (prop,value) = G.getProperties(e);
        ES'.insert(e');
        EP'.insert(e',prop,value);
    FOR EACH edge e=(N,C)^T outgoing N in G DO
        e' = (N',C)^T;
        (prop,value) = G.getProperties(e);
        ES'.insert(e');
        EP'.insert(e',prop,value);
    removeSets(G,{N},{},{},{});
    addSets(G,{N'},ES',NP',EP');
END
```

```
PROCEDURE replaceSubGraph (G: QualityGraph, NS: NodeSet, ES: EdgeSet, NS': NodeSet,
                           ES': EdgeSet, NP': NodePropertySet, EP': EdgePropertySet)
    removeSets(G,NS,ES,{},{});
    addSets(G,NS',ES',NP',EP');
END
```

```
PROCEDURE decomposeNode (G: QualityGraph, N: Node, NS': NodeSet, ES': EdgeSet,
                         NP': NodePropertySet, EP': EdgePropertySet)
    removeSets(G,{N},{},{},{});
    addSets(G,NS',ES',NP',EP');
END
```

```
PROCEDURE parallelizeNodes (G: QualityGraph, NS: NodeSet, ES': EdgeSet, EP':
                            EdgePropertySet)
    EdgeSet ES;
    FOR EACH edge e incoming a node of NS in G DO
        ES.insert(e);
    FOR EACH edge e outgoing a node of NS in G DO
        ES.insert(e);
    removeSets(G,{},ES,{},{});
    addSets(G,{},ES',{},EP');
END
```

```
PROCEDURE changeNodesProperties (G: QualityGraph, NP': NodePropertySet)
    addSets(G,{},{},NP',{});
END
```

```
PROCEDURE changeEdgesProperties (G: QualityGraph, EP': EdgePropertySet)
    addSets(G,{},{},{},EP');
END
```

Algorithm 3.8 – Macro improvement actions

In the remaining of the sub-section, we discuss the use of macro improvement actions according to the four objectives above mentioned.

Reducing processing costs

A typical way of reducing processing costs is replacing the current implementation of an activity by a more performing one. To this end, a new ad-hoc process can be created or an existent component can be reused (e.g. a web service provided by a third party developer). In the latter case, additional activities (adapters) may be necessaries for connecting the existing activities to the new component. The cost of adapters must be taken into account. Complex activities may be decomposed in simpler portions in order to replace only some of them.

Another improvement action consists in parallelizing the execution of some activities. Analogously, additional activities may be necessaries for controlling the execution (e.g. waiting for the end of all the activities before executing successors).

Further actions include running some activities in a more performing server and powering CPU and memory. In this case, the topology of the quality graph does not change but property values are modified to reflex the new scenario (the processing cost may be one of the updated properties). In addition, specific actions can be defined for specific scenarios.

Example 3.19. Consider activity A_6 of the quality graph of Figure 3.19a, which integrates data coming from predecessor activities and also builds aggregates and statistics needed by successor activities. The process that implements A_6 can be decomposed in three routines (Figure 3.19b): activity A_{61} performs data integration, activity A_{62} computes statistics for successor A_7 and activity A_{63} build aggregates for successor A_8. The decomposition allows replacing some routines by more performing ones, e.g. replacing A_{61} by A_{64} (Figure 3.19c) and allocating more resources to it in order to execute it more performingly. Note that the time for building statistics for A_7, is unnecessarily paid for data going to A_8, thus, activities A_{62} and A_{63} can be parallelized (Figure 3.19c).

The applied improvement actions are:

- decomposeNode (G, A_6, {A_{61},A_{62},A_{63}}, {(A_5,A_{61}),(A_4,A_{61}),(A_{61},A_{62}),(A_{62},A_{63}),(A_{63},A_7),(A_{63},A_8)},
  {<A_{61},'ProcessingCost',3>,<A_{62},'ProcessingCost',2>,<A_{63},'ProcessingCost',1>},
  {<(A_5,A_{61}),'InterProcessDelay',12>,<(A_4,A_{61}),'InterProcessDelay',0>,
  <(A_{61},A_{62}),'InterProcessDelay',0>,<(A_{62},A_{63}),'InterProcessDelay',0>,
  <(A_{63},A_7),'InterProcessDelay',7>,<(A_{63},A_8),'InterProcessDelay',7>})
- replaceNode (G, A_{61}, A_{64}, {<A_{64},'ProcessingCost',2>})
- changeNodesProperties (G, {<A_{64},'ProcessingCost',1>})
- parallelizeNodes (G, {A_{62},A_{63}}, {(A_{64},A_{62}),(A_{64},A_{63}),(A_{62},A_7),(A_{63},A_8)},
  {<(A_{64},A_{62}),'InterProcessDelay',0>,<(A_{64},A_{63}),'InterProcessDelay',0>,
  <(A_{62},A_7),'InterProcessDelay',7>,<(A_{63},A_8),'InterProcessDelay',7>}) □

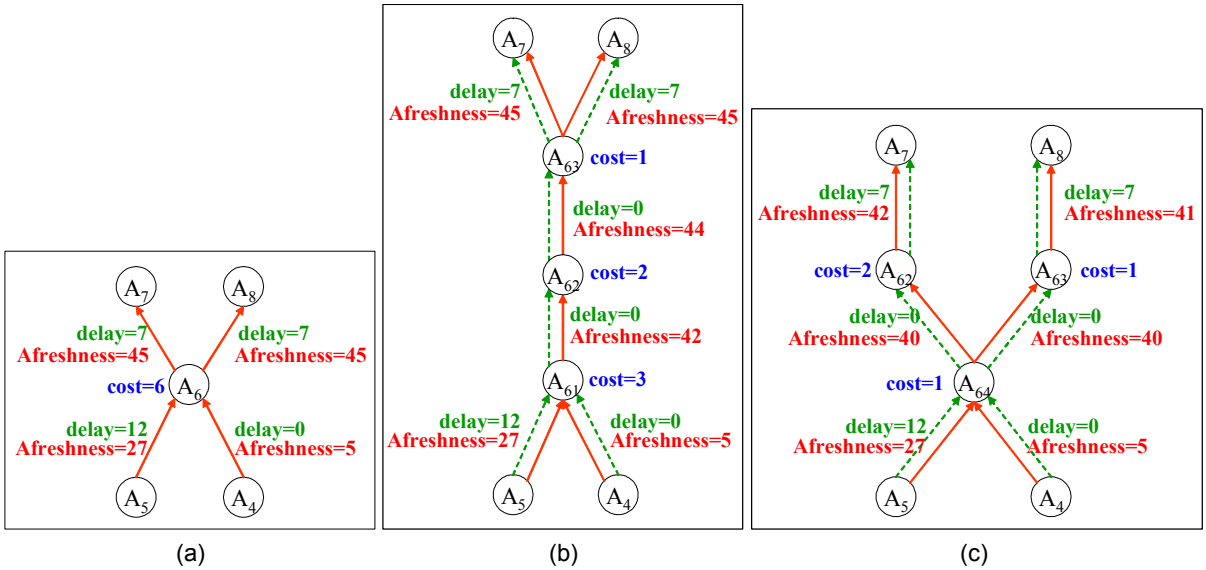

Figure 3.19 – Portions of a quality graph: (a) before performing improvement actions, (b) after decomposing an activity, (c) after replacing and parallelizing activities

Reducing inter-process delays

A typical way of reducing inter-process delays is synchronizing activities, eventually changing their synchronization policies, for executing one after the other. Activities may have synchronous or asynchronous (pull or push) policies, driven by temporal or non-temporal events (see Sub-section 2.3 of Chapter 2) and coexistence of different policies may introduce delays. Even synchronization-related properties (synchronization policies, execution frequencies, synchronization events, control events) are inherent to control flow, they indirectly causes inter-process delays among activities. Synchronizing activities do not necessarily mean that activities must execute one immediately after the other; certain delays may be necessaries due to processing constraints, for example the need of sequentially execute other activities or system routines. Then, improvement actions may reduce delays instead of eliminating them.

Unfortunately, improving the synchronization among some activities may worsen the synchronization with another ones. Consequently, when applying local synchronization techniques to portions of the quality graph, the impact to the whole graph should be studied. Next example illustrates this fact.

> **Example 3.20.** Figure 3.20a shows three activities A_5, A_6 and A_8, the two former with periodic pull policies and the latter with aperiodic pull policy. Activity A_5 executes every 12 units of time and activity A_6 every 7 units of time, both materializing data. As they execute asynchronously, activity A_6 reads data that have been materialized for some time, 12 units of time in the worst case. Activity A_8 executes aperiodically, when users pose queries, but also reads data that has been previously materialized, 7 units of time in the worst case. Figure 3.20b illustrates their execution over time (executions are represented by rectangles, which lengths indicate processing costs) and the materialized data that is used as input (dotted lines).
>
> If activity A_6 is synchronized with activity A_5 (its execution frequency is changed for executing every 12 units of time, just after A_5) inter-process delays are negligible. But note that in this case, the inter-process delay with activity A_8 will be 12 units of time in the worst case (instead of 7).
>
> However, if we change the execution frequency of activity A_6 for executing every 6 units of time (as shown in Figure 3.20c), inter-process delays between activities A_5 and A_6 are either 0 or 6 units of time (6 units of time in the worst case, instead of 12) and delays between activities A_6 and A_8 are at most 6 units of time (instead of 7).

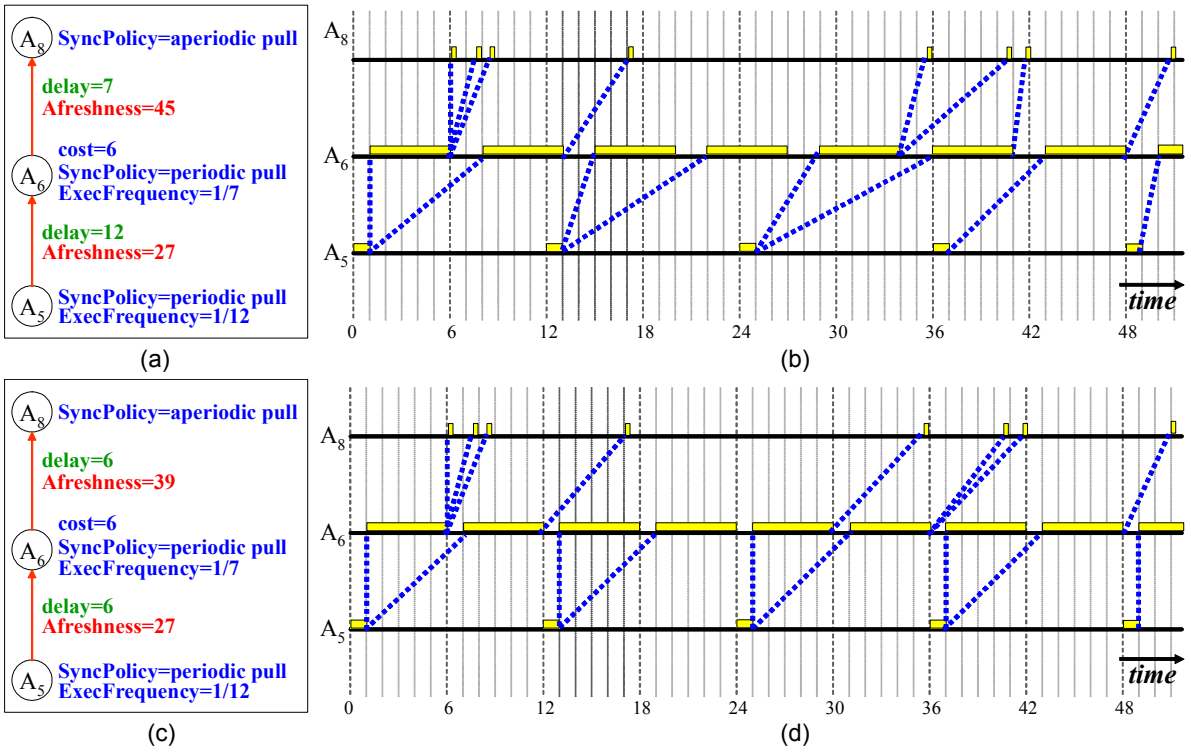

Figure 3.20 – Synchronization of activities: (a) portion of a quality graph, and (b) diagram of activity executions before the improvement action, and (c) portion of the quality graph and (d) diagram of activity executions after the improvement action

The applied improvement actions are:

- changeNodesProperties (G,{<A_6,'ExecFrequency',1/6>})
- changeEdgesProperties (G,{<(A_5,A_6),'InterProcessDelay',6>,<(A_6,A_8),'InterProcessDelay',6>}) □

Increasing the refreshment frequency of materialized data (i.e. executing the activity more frequently) may improve freshness, as shown in previous example. However, note that refreshment frequencies are constrained to allowed source accesses and activity processing times. The former arises when sources can only be queried at certain times or the number of source accesses is limited (for example because of source access price) and requires negotiation with source data providers for relaxing access constraints. The latter is intuitive: if an activity last 5 units of time for executing, it cannot execute every 2 units of time. So, synchronization techniques may be complemented by techniques for reducing processing costs, for example, decomposing an activity in portions with smaller processing costs and then increasing the refreshment frequency of new activities.

In previous example we only had two activities with periodic execution. The problem is more complex when activities have several predecessors and successors, each one with different synchronization policies. Furthermore, the combination of pull and push policies with different types of events makes very difficult the development of general synchronization techniques; specific techniques should be studied for particular application scenarios. In Section 5, we present a detailed analysis for one concrete scenario.

Reducing source actual freshness

If a source provides with data having too high freshness actual values, it can be substituted by another source providing with the same type of data but having lower freshness actual values. Note that the decision of substituting a source may also depend on other quality factors (e.g. accuracy, completeness, availability). The replacement of a source may imply the modification of other DIS components, especially wrapper activities.

Note that the new source may provide with incomplete information, for example, if it does not provide with certain attributes (which may be provided by another source). Then, a source can be replaced by a sub-graph (accessing several sources) that calculates the same data. Note that even source data actual freshness may be reduced, processing costs of new activities may be higher, so they should also be studied. Analogously, certain sub-graphs representing the access to several sources can be replaced by the access to a unique source.

Example 3.21. Figure 3.21a shows a portion of a quality graph accessing to source S_2. Figure 3.21b shows the replacement of source S_2 for sources S_4 and S_5, and the consequent replacement of wrapper activity A_2 for activities A_{21}, A_{22} and A_{23}. The applied improvement action is:

- replaceSubGraph (G, {S_2,A_2}, {}, {S_4,S_5,A_{21},A_{22},A_{23}}, {(S_4,A_{21}),(S_5,A_{22}),(A_{21},A_{23}),(A_{22},A_{23}),(A_{23},A_5)},
  {<S_4,'SourceActualFreshness',12>,<S_5,'SourceActualFreshness',6>,<A_{21},'ProcessingCost',2>,
  <A_{22},'ProcessingCost',1>,<A_{23},'ProcessingCost',2>},
  {<(A_{21},A_{22}),'InterProcessDelay',0>,<(A_{22},A_{23}),'InterProcessDelay',0>,
  <(A_{23},A_5),'InterProcessDelay',0>}) □

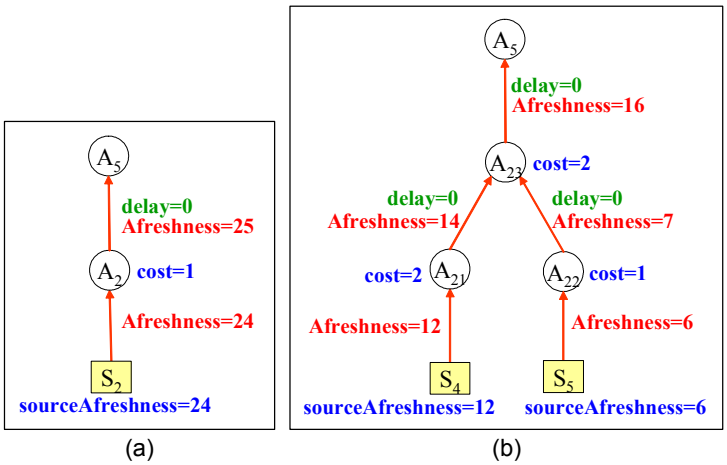

Figure 3.21 – Portions of the example quality graph (a) before and (b) after replacing a source

The comparison of different source data providers allows selecting the sources that provide the freshest data. The propagation of freshness expected values aids in such selection. Negotiating with source providers is also possible, demanding (and eventually paying) for a better service (i.e. better source data actual freshness).

Augmenting target expected freshness

Finally, when data freshness cannot be further improved, i.e. user expectations are too high for the data that can be effectively obtained from data repositories or improvement actions are too expensive (e.g. imply the acquisition of new hardware), we should negotiate with users in order to relax freshness expectations. Furthermore, many users have incremental behaviors, i.e. they ask for certain freshness values (very exigent) and if the DIS cannot provide these values, they try with relaxed values and so on.

> **Example 3.22.** Consider that a user has demanded a freshness expected value of 10 units of time for certain data target T but the DIS cannot convey so fresh data, so the user is notified receiving no data. Then, the user decides to relax freshness expectations demanding a freshness expected value of 15 units of time and then, he tries his query again. The applied improvement action is:
>
> - changeNodesProperties (G, {<T,'TargetExpectedFreshness',15>}) □

Summary

All previously discussed actions are general enough to be applied to different types of DIS but their use for freshness enforcement should be guided by some high-level strategy in order to be effective. On the contrary, the ad-hoc use of improvement actions may be not viable. For example, manually finding optimal refreshment frequencies in order to minimize inter-process delays, in a DIS with several tens of activities is not an easy matter. Our approach consists in studying the DIS in a high-level quality graph and zooming in the critical paths (or portions of the paths) in concentrate improvement actions in the nodes and edges of the path. Clearly, in some scenarios, the actions applied to critical paths are not enough for enforcing data freshness and other portions of the graph should be studied, for example, when the synchronization of some activities degraded the synchronization of other ones. The methods for browsing in the hierarchy of quality graphs are useful to this end.

The set of macro actions described in this sub-section is not complete, in the sense of trying to cover all possible improvement strategies. On the contrary, our approach allows the definition of specialized strategies for concrete application scenarios, as the one that will be presented in Section 5.

Next sub-section summarizes our approach with an example.

4.5. Summarizing example

In the following example, we analyze a quality graph where freshness expectations are not achieved. We firstly compute the critical path and we zoom in activities with higher processing costs and synchronization delays, analyzing possible improvement actions.

> **Example 3.23.** Consider the DIS of Figure 3.22a, which has two data sources ($Source_1$ and $Source_2$) and two data targets ($Query_1$ and $Query_2$). Figure 3.22b shows a first zoom+ operation in order to show the activities that compose the DIS process, which perform the data extraction ($Extr_1$ and $Extr_2$), integration ($Integ_3$) and aggregation ($Aggr_4$ and $Aggr_5$). Considering the properties shown in Figure 3.22b, we achieve freshness values of 68 hours for $Query_1$ and 61 hours for $Query_2$. The former is acceptable, but the latter is too high and must be reduced in order to achieve user expectations.
>
> We analyze the critical path for $Query_2$, i.e. [$Source_2$, $Extr_2$, $Integ_3$, $Aggr_5$]. First of all, we browse in the hierarchy of quality graphs, focusing in each activity of the critical path and zooming in them in order to have an overview of candidate portions to improve, as shown in Figure 3.22 (c, d and e). $Extr_2$ extracts and cleans information from $Source_2$; it is composed of three sub-activities: the wrapping process ($Wrap_{21}$) and two cleaning processes ($Clean_{22}$ and $Clean_{23}$). $Integ_3$ consolidates data extracted from both sources; it is composed of an initial integration activity ($Integ_{31}$), a cleaning activity that corrects some kinds of common errors ($Clean_{32}$) and a final activity that performs complex calculations ($Calc_{33}$). Due to the complexity of operations, both $Integ_{31}$ and $Calc_{33}$ materialize data once a day. $Aggr_5$ builds data needed by $Query_2$; it is composed of an aggregation activity ($Aggr_{51}$) and a posterior cleaning process for grouped data ($Clean_{52}$).

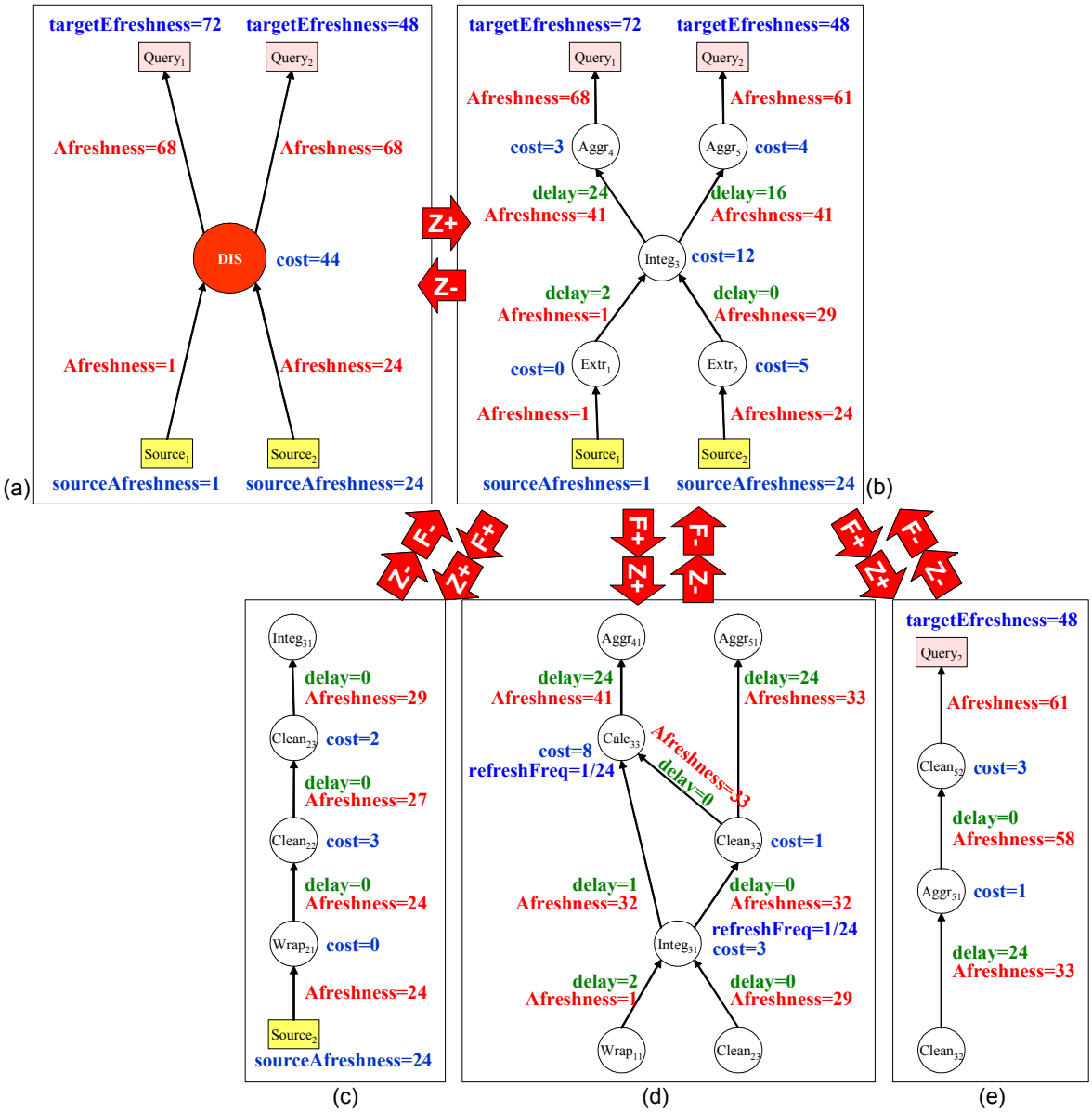

Figure 3.22 – Browsing in the critical path for $Query_2$: (a)highest-level quality graph, (b) zoom+ in the DIS node; and focus+ and zoom+ in activities: (c) $Extr_2$, (d) $Integ_3$, and (e) $Aggr_5$

Let's start analyzing $Integ_3$ and its synchronization with successors. Note that there is a big inter-process delay between $Clean_{32}$ and $Aggr_{51}$ due to the refreshment frequencies of $Integ_{31}$, as shown in Figure 3.22d. A good improvement action consists in increasing the refreshment frequency of $Integ_{31}$ (and thus executing $Calc_{33}$ more frequently). This action should be negotiated with the provider of $Source_2$ (to know if the source can be accessed more frequently) and DIS administrators (to know if the wrapper can be executed more frequently). Note that as activity Calc33 is very costly, it may continue executing once a day. Then, the first improvement actions consists in changing the refreshment frequency of $Integ_{31}$ to 12 hours and consequently changing the inter-process delay between $Clean_{32}$ and $Aggr_{51}$ to 12 hours, i.e.:

- changeNodesProperties (G_3,{<$Integ_{31}$,'RefreshFrequency',1/12>})
- changeEdgesProperties (G_3,{<($Clean_{32}$,$Aggr_{51}$),'InterProcessDelay',12>)

Figure 3.23a shows the detailed graph for activity $Integ_3$ after these actions. Returning to the summarized graph, freshness actual value for $Query_2$ is 49 hours, which is still not acceptable.

Another improvement action consists in parallelizing the cleaning sub-activities of $Extr_2$. Both activities act over different data so the output of the wrapper can be decomposed in two disjoint sets (substituting $Wrap_{21}$

by $Wrap_{24}$) and merged at the end (adding activity Mer_{25}), both having negligible cost. As $Clean_{23}$ finishes before $Clean_{22}$, it materializes data. Actions are:

- replaceSubGraph (G, {}, {($Clean_{23}$,$Integ_{31}$)}, {Mer_{25}}, {($Clean_{23}$,Mer_{25}),(Mer_{25},$Integ_{31}$)},
  {<Mer_{25},'ProcessingCost',0>},
  {<($Clean_{23}$,Mer_{25}),'InterProcessDelay',0>,<(Mer_{25},$Integ_{31}$),'InterProcessDelay',0>})
- replaceNode (G, $Wrap_{21}$, $Wrap_{24}$, {<$Wrap_{24}$,'ProcessingCost',0>})
- parallelizeNodes (G_3, {$Clean_{22}$,$Clean_{23}$},
  {($Wrap_{24}$,$Clean_{22}$),($Wrap_{24}$,$Clean_{23}$),($Clean_{22}$,Mer_{25}),($Clean_{23}$,Mer_{25})},
  {<($Wrap_{24}$,$Clean_{22}$),'InterProcessDelay',0>,<($Wrap_{24}$,$Clean_{23}$),'InterProcessDelay',0>,
  <($Clean_{22}$,Mer_{25}),'InterProcessDelay',0>,<($Clean_{23}$,Mer_{25}),'InterProcessDelay',1>})

Figure 3.23b shows the detailed graph for activity $Extr_2$ after these actions. Returning to the summarized graph (Figure 3.23c), freshness actual value for $Query_2$ is 47 hours, which is acceptable. □

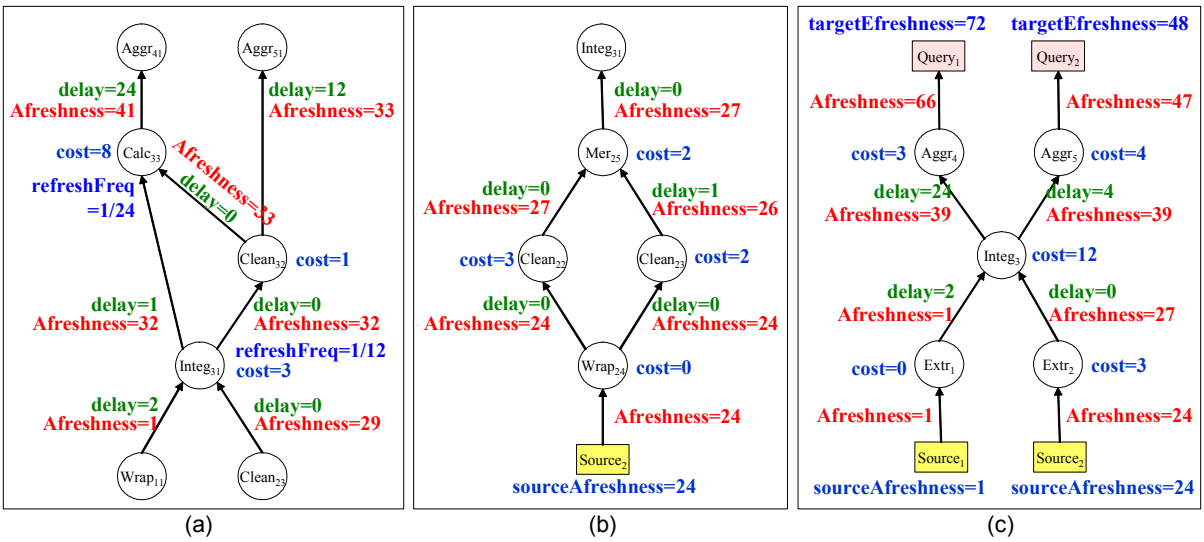

Figure 3.23 – Quality graphs after improvement actions: (a) detailed graph for $Integ_3$, (b) detailed graph for $Extr_2$, and (c) summarized graph (after all actions)

Along this section we have illustrated that data freshness enforcement may consist of several improvement actions (we have given examples of actions), which use for DIS design or reengineering depends on the particularities of concrete application scenarios. To illustrate this, next section fixes an application scenario and analyzes one improvement strategy: the synchronization of activities in order to enforce data freshness. Other strategies can be analyzed analogously for specific application scenarios; the quality evaluation framework and the general strategies discussed in this section (critical path, top-down analysis, actual and expected freshness propagation) may help in the analysis.

5. Synchronization of activities

Previous section proposed a set of elementary and macro improvement actions and argued that improvement strategies for DIS design can be enunciated for concrete DIS scenarios based on improvement actions. The purpose of this section is to illustrate the development of an improvement strategy.

Based on an improvement action (*change of execution frequencies of activities*) suggested in Sub-section 4.4 for reducing inter-process delays, we discuss the determination of appropriate execution frequencies for activities. We consider a concrete application scenario where activities have specific synchronization policies and we deal with their synchronization in order to find the optimal execution frequencies that allow achieving freshness expectations. Next sub-section motivates and states the problem and the rest of the section discusses optimal and heuristic solutions.

5.1. DIS synchronization problem

The comparison among source data actual freshness and target data expected freshness serves to determine constraints for DIS design. Their difference, if positive, indicates the longest period of time that DIS processes are allowed to spend in manipulating data, i.e. it is an upper bound for the execution delay of the DIS, which includes the processing costs of activities and the inter-process delays among them. If processing costs of activities are not too high, i.e. if all activities can execute in less time than the bound for the execution delay, the achievement (or overdraw) of freshness expectations depends on inter-process delays. The idea is to synchronize activities in a way that inter-process delays are sufficiently small to allow achieving freshness expectations.

> **Example 3.24.** Consider the quality graph of Figure 3.24. In the path $[S_2,A_2,A_3,A_4,T_1]$, the difference between freshness expectations and source data actual freshness is 12 units of time. The activities in the path summarize 4 units of time of processing costs, so 8 units of time can be spent in inter-process delays, i.e. greater values will cause freshness actual value to overdraw freshness expected value.
>
> If activity A_3 executes every 3 units of time and activity A_4 executes asynchronously with each user query, the delay among A_3 and A_4 will be 3 units of time in the worst case. Analogously, we can set the execution frequency of A_2 for obtaining a delay of 5 units of time between A_2 and A_3. These delays of 3 and 5 units of time can be tolerated. Of course, delays in the path $[S_1,A_1,A_3,A_4,T_1]$ must also be analyzed. □

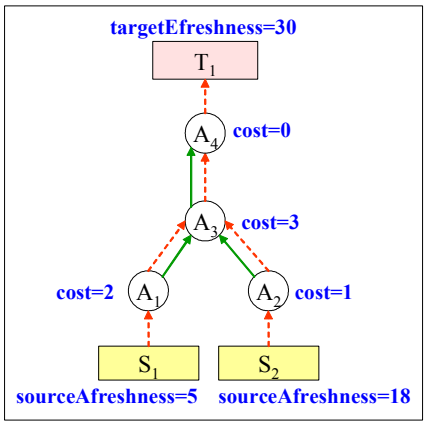

Figure 3.24 – Determination of execution policies

Before enunciating the problem, we state some hypotheses that fix the application scenario, specially, the DIS synchronization policies:

- Activities providing data to targets (called conveyance activities) execute when users ask for data (pulled by data targets) and do not materialize data; the other activities (called intermediate activities) execute asynchronously, guided by periodic pull events and may materialize data.
- Activities must finish execution before starting a new execution. Consequently, the execution period* of an activity should be greater or equal to its processing cost.
- Inter-process delays are determined exclusively by the execution frequencies of activities, i.e. they are not influenced by other DIS properties (e.g. hardware constraints, scheduling restrictions, communication delays). Activities can be executed in parallel. Control flow coincides with data flow.
- Sources present no access constraints, i.e. they can be accessed as frequently as needed.
- Processing costs, source data actual freshness and target data expected freshness, as well as execution periods, are all integer values, multiples of certain unit of time *T* (e.g. a day, an hour, fifteen minutes). The combineActualFreshness function returns the maximum of input values.

Synchronizing activities consist in finding an appropriate execution frequency for each intermediate activity in order to coordinate the whole DIS and obtain inter-process delays that allow achieving freshness expectations. Note that different synchronization policies (providing the execution frequency for each activity) may allow

* The execution period is the inverse of the execution frequency, i.e. if an activity executes every *P* units of time, execution frequency is *1/P* and execution period is *P*.

achieving freshness expectations. We need a method for finding feasible policies but we also need to decide which one to choose. Several criteria for selecting when a policy is better than another can be proposed. An example of this kind of criteria is minimizing the overall maintenance cost, which can be calculated as the processing cost of each activity, weighted par its execution frequency [Theodoratos+1999]. Finding the best synchronization of activities is an optimization problem. It can be stated as follows:

Definition 3.12 Given a quality graph G, labeled with source data actual freshness, target data expected freshness and processing cost properties, the *DIS synchronization problem* consists in finding the most appropriate execution frequencies for intermediate activities in a way that inter-process delays allow achieving freshness expectations, minimizing maintenance cost. □

In the following sub-sections we characterize the solution space and propose some algorithms to solve the problem.

5.2. Characterization of the solution space

A solution to the DIS synchronization problem is a set of execution periods (or execution frequencies), one for each intermediate activity. In this sub-section we characterize the solution space, determining the conditions that a solution must verify.

Basically, the idea is to bound the amount of time that can be consumed in synchronization, studying the difference between actual and expected freshness. We firstly obtain upper and lower bounds for the freshness of the data produced by each activity, which are called *uppermost* and *lowest* freshness values respectively. The lowest freshness value is calculated propagating source data actual freshness, from sources to targets, adding the processing costs of activities. This is the lowest freshness value that the DIS might obtain, which is achieved if all activities are synchronized for starting as soon as the predecessor activities finish, without data materialization and consequently, without inter-process delays. Note that this kind of synchronization is not always possible. The uppermost freshness value is calculated propagating target data expected freshness, from targets to sources, subtracting the processing costs of activities. This is the greatest freshness value that can be supported by the DIS in order to achieve freshness expectations for data targets. The lowest and uppermost freshness values can be calculated using the propagation algorithms described in Section 3 (see Algorithm 3.1 and Algorithm 3.2) overloading the getInterProcessDelay function in order to return zero.

If for some activity, the lowest freshness value is greater than the uppermost freshness value, freshness cannot be assured no matter the synchronization of activities and other improvement actions should be followed, for example, for reducing processing costs. On the contrary, if the uppermost freshness value is greater than the lowest freshness value, their difference is an upper bound for the execution period of the activity. We define the *greatest execution period* for the activity as such difference.

Definition 3.13 The *greatest execution period* of an activity is calculated as the difference between uppermost and lowest freshness values. □

The valid execution periods for an activity are those comprised among the processing cost (smallest execution period that can be implemented) and the greatest execution period (calculated as explained before). Obviously, if for some activity, the processing cost is greater than the greatest execution period, freshness cannot be assured no matter the synchronization of activities and consequently, other improvement actions should be followed.

We can now characterize the solution space.

Definition 3.14 Given a quality graph G, with k intermediate activities $\{A_1 \ldots A_k\}$ that produce intermediate data consumed by other activities and n-k conveyance activities $\{A_{k+1} \ldots A_n\}$ that deliver data to some target node, *the solution space of the DIS synchronization problem* consists of two conditions:

(i) $\text{ProcessingCost}(A_i) \leq \text{ExecutionPeriod}(A_i) \leq \text{GreatestExecutionPeriod}(A_i)$, $1 \leq i \leq k$

(ii) $\text{ActualFreshness}(A_i, T_i) \leq \text{ExpectedFreshness}(A_i, T_i)$, $k+1 \leq i \leq .. n$, for T_i being the successor of A_i

The former condition ranges the execution period of intermediate activities (which are the variables of the problem) between the processing cost and the greatest execution period of the activity. The latter assures that actual freshness is not greater than expected freshness for all conveyance activities. □

For calculating data freshness (in order to check the second condition), inter-process delays must be calculated. Let's start defining the notion of synchronism.

Definition 3.15 Two activities are *synchronized* with a shift time of K (*K-synchronized* for short), if for each execution of the latter activity there is an execution of the former exactly K units of time before. □

Example 3.25. Consider two activities A and B, where A executes every 4 hours. If B also executes every 4 hours, just 1 hour after A, then they are 1-synchronized. If B executes every 8 hours, immediately after an execution of A, they are 0-synchronized. But if B executes every 3 hours, it is shifted with A in a variable number of hours, between 0 and 3 hours, so they are not synchronized. □

If two activities A and B are 0-synchronized, B can start executing as soon as A finishes, which will minimize the waste of time between them. The inter-process delay among them is negligible. If two activities A and B are not 0-synchronized, A must materialize data, which will be asynchronously read by B. In this case, there is a positive inter-process delay between A and B. In order to estimate the inter-process delays, we analyze the execution periods of activities. If A and B are both periodically executed, the inter-process delay among them can be calculated, in the worst case, with the following formula:

InterProcessDelay(A,B) = ExecutionPeriod(A) – GCD(ExecutionPeriod(A),ExecutionPeriod(B))

where GCD is the greatest common divisor function. Note than when A and B are 0-synchronized, the GCD function equals the execution period of A, so the inter-process delay is zero. Also note that the inter-process delay obtained with this formula should be increased if the execution of activities is shifted, for example, for executing B one hour after. If B is not periodically executed (which is the case of conveyance activities) the inter-process delay between A and B can be calculated, in the worst case, with the following formula:

InterProcessDelay(A,B) = ExecutionPeriod(A)

Example 3.26. Figure 3.25 shows the three synchronization cases discussed in Example 3.25. In case (b), GCD(4,8)=4 so inter-process delay is 4–4=0. In case (c), GCD(4,3)=1, so inter-process delay is 4-1=3 in the worst case; Figure 3.25c illustrates the cases where inter-process delay takes the maximum value. In case (a), GCD(4,4)=4, so inter-process delay is 4–4=0; however, for external reasons activity B is configured for executing one hour after A, 1-synchronized, so delay is 1. □

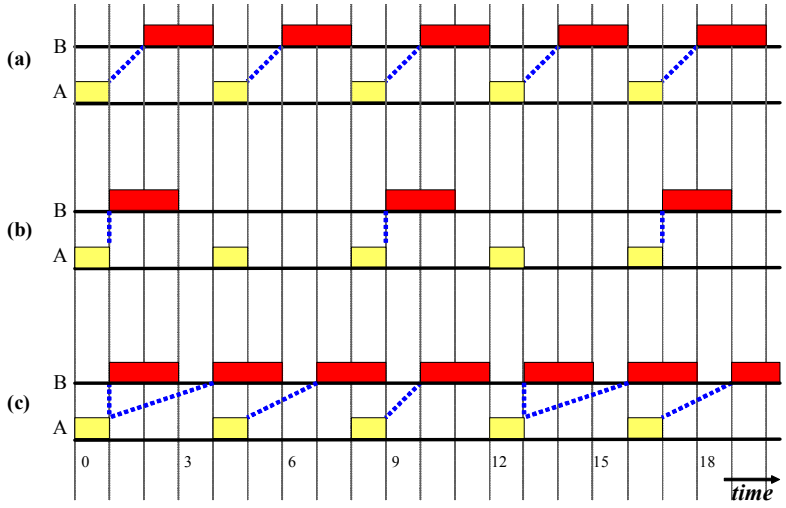

Figure 3.25 – Determination of inter-process delays

Having a calculation method for inter-process delays, the getInterProcessDelay function can be conveniently overloaded and actual freshness can be evaluated using the ActualFreshessPropagation algorithm, which allows to practically check the second condition of the solution space (see Definition 3.14). However, in order to formalize the problem, the second condition should be expressed in terms of the variables (execution periods). As stated in the corollary of Lemma 3.3, the freshness of the data produced by an activity node coincides with the path freshness of its critical path. If we calculate all paths from a source to a conveyance activity, we can

decompose the second condition in a set of conditions, one for each path, stating that the path freshness must be lower or equal to expected freshness.

We formalize the problem as a nonlinear integer programming (NLIP) problem (see Definition 3.16). The variables (x_i) represent the execution periods of intermediate activities. The objective function corresponds to the overall maintenance cost (sum of processing costs weighted by execution frequencies), which must be minimized. The constraints delimit the solution space. The former ranges variables between the processing cost and the greatest execution period of the activity. The latter states that for each path going from a source node to a conveyance activity, the path freshness must be lower or equal to the expected freshness of the activity. This assures that actual freshness is not greater than expected freshness for all conveyance activities. Note that neither the objective function nor the second constraint (due to GCD) are linear.

Definition 3.16 Given a quality graph G, with m sources $\{S_1 \ldots S_m\}$, k intermediate activities $\{A_1 \ldots A_k\}$ and n-k conveyance activities $\{A_{k+1} \ldots A_n\}$, let c_i, z_i, af_i, ef_i be non-negative integers corresponding to:

- c_i: the processing cost of activity A_i, $1 \le i \le n$,
- z_i: the greatest execution period of activity A_i, $1 \le i \le k$.
- af_i: the source data actual freshness of source node S_i, $1 \le i \le m$,
- ef_i: the expected freshness for data produced by activity A_i, $k < i \le n$.

Let $\{P_1 \ldots P_r\}$ be the set of paths from a source node to a conveyance activity. P_j has the form $[P_{j0}, P_{j1}, \ldots P_{jw_j}, P_{j(w_j+1)}]$, $1 \le j \le r$, $w_j \le k$, where:

- P_{j0} is the index of a source relation, i.e. $1 \le P_{j0} \le m$,
- P_{ji} is the index of a intermediate activity, i.e. $1 \le P_{ji} \le k$, $1 \le i \le w_j$,
- $P_{j(w_j+1)}$ is the index of a conveyance activity, i.e. $k < P_{j(w_j+1)} \le n$.

The *DIS synchronization problem* is formalized as follows:

Minimize: $\sum_{i=1..k} (c_i / x_i)$

subject to:

$c_i \le x_i \le z_i$, i:1..k

$af_{P_{j0}} + \sum_{i=1..(w_j+1)} (c_{P_{ji}}) + \sum_{i=1..(w_j-1)} (x_{P_{ji}} - GCD(x_{P_{ji}}, x_{P_{j(i+1)}})) + x_{P_{jw_j}} \le ef_{P_{j(w_j+1)}}$, i:1..k, j:1..r □

Example 3.27. Consider the quality graph of Figure 3.26. Maximal execution periods are calculated propagating uppermost and lowest freshness values. There are two paths from sources to conveyance activities: $[S_1,A_1,A_3,A_4]$ and $[S_2,A_2,A_3,A_4]$. The NLIP problem for this quality graph is:

Minimize: $2/x_1 + 1/x_2 + 3/x_3$

subject to:

$2 \le x_1 \le 20$

$1 \le x_2 \le 8$

$3 \le x_3 \le 8$

$x_1 + x_3 - GCD(x_1,x_3) \le 20$

$x_2 + x_3 - GCD(x_2,x_3) \le 8$

The two latter conditions have been simplified from:

$5 + 2 + 3 + 0 + x_1 - GCD(x_1,x_3) + x_3 \le 30$

$18 + 1 + 3 + 0 + x_2 - GCD(x_2,x_3) + x_3 \le 30$ □

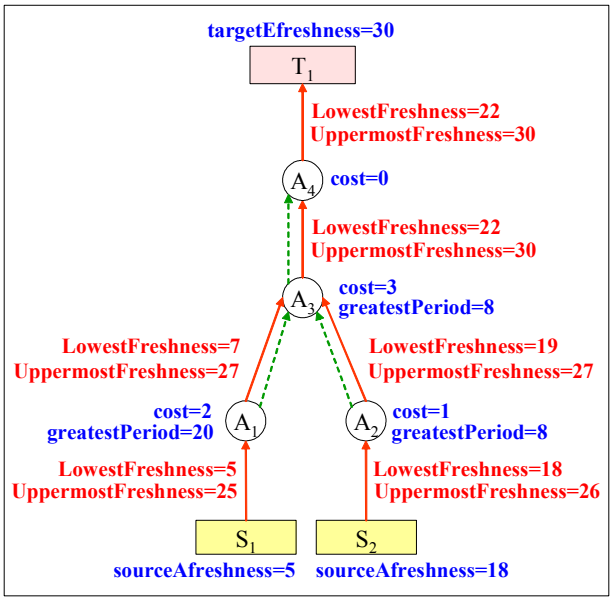

Figure 3.26 – Quality graph labeled with uppermost and lowest freshness values and greatest execution periods

In practice, using the ActualFreshnessPropagation algorithm for obtaining freshness actual values is better than generating all possible paths from sources to conveyance activities. The following sub-section discusses different algorithms for solving the problem.

5.3. Solutions to the DIS synchronization problem

In this sub-section we discuss different solutions to the DIS synchronization problem. We first present a naïve method for rapidly finding a solution (generally non optimal). Then, we discuss a branch-and-bound* algorithm for finding the optimal solution. The solution built with the naïve method is used for pruning the solution space. Other properties are analyzed for providing further pruning. However, as most branch-and-bound methods, the algorithm only can be executed with small size graphs due to its high complexity. We then discuss heuristics for finding non-optimal but good-enough solutions.

All algorithms return a tuple (an array of execution periods, one for each intermediate activity) that belongs to the solution space, i.e. verifies the two constraints of the problem defined in previous sub-section. Each tuple has associated a maintenance cost, i.e. its value for the objective function. Tuples are called *candidate tuples* when feasibility is not yet checked and *solutions* when they are a feasible solution to the problem.

We define the following type to manage tuples and their costs:

```
TYPE Tuple = RECORD (periods: ARRAY OF INTEGER, cost: FLOAT)
```

Next sub-sections describe the algorithms.

Naïve algorithm

The naïve algorithm builds a candidate tuple, assigning the same execution period to all the intermediate activities, in a way that all intermediate activities are 0-synchronized without inter-process delay among them. The unique inter-process delays are those with conveyance activities. The value assigned to all variables is the smallest of the greatest execution periods of activities. Observe that if for an activity, its processing cost is greater than the assigned execution period, then, the candidate tuple is not a feasible solution (it does not verify the first problem constraint). So, the naïve algorithm does not always find a solution.

A pseudocode of the naïve algorithm can be sketched as shown in Algorithm 3.9 (*BaseSolution* function). It first obtains the minimum of the greatest execution periods and then assigns it as execution period of all intermediate activities. If the candidate tuple is not a feasible solution, it returns an infinite cost tuple. The MaintenanceCost function calculates the objective function.

```
FUNCTION BaseSolution (G: QualityGraph) RETURNS Tuple
   Tuple T;
   P = min {G.getPropertyValue(A,"GreatestExecutionPeriod") / A is an intermediate activity}
   FOR EACH intermediate activity A in G DO
      IF (getProcessingCost(G,A) > P) THEN
         T.cost = infinite;
         RETURN T;
      ELSE
         T.periods[A] = P;
   ENDFOR;
   T.cost = MaintenanceCost (G,T);
   RETURN T;
END
```

Algorithm 3.9 – Naïve algorithm for building a base solution

* Branch-and-bound is a classical method for solving NLIP problems [Cooper 1981] [Li+2003].

If the naïve algorithm finds a solution, such solution can be used as first solution for an exhaustive branch-and-bound algorithm (for pruning some tuples), if not, we take the infinite cost tuple (that obviously will not prune any tuple). Next sub-section presents a branch-and-bound algorithm for finding an optimal solution.

Optimal algorithm

The optimal algorithm will traverse the solution space, exhaustively testing all possible values for variables (comprised between the range imposed by the first problem constraint, i.e. the processing cost and the greatest execution period). The traversal starts with a tuple composed of the greatest execution periods and proceeds, in each iteration, decreasing one of the variables in a unit of time, backtracking when we can assure that the optimal solution cannot be found decreasing more variables (pruning criteria).

```
FUNCTION OptimalSolution (G: QualityGraph) RETURNS Tuple
   Tuple Best = BaseSolution (G);
   Tuple T;
   FOR EACH intermediate activity A in G DO
      INTEGER P = G.getPropertyValue(A,"GreatestExecutionPeriod");
      G.setPropertyValue(A,"ExecutionPeriod",P);
      T.periods[A] = P;
   ENDFOR;
   T.cost = MaintenanceCost (G,T);
   RETURN BacktrackingIteration (G,T,Best);
END
```

```
FUNCTION BacktrackingIteration (G: QualityGraph, T: Tuple, Best: Tuple) RETURNS Tuple
   IF (T.cost < Best.cost) THEN
      G = ActualFreshnessPropagation (G);
      IF IsFeasibleSolution (G,T) THEN RETURN T;
      ELSE
         FOR EACH intermediate activity A in G DO
            Tuple S = T;
            IF (S.periods[A] > getProcessingCost(G,A)) THEN
               S.periods[A] --;
               G.setPropertyValue(A,"ExecutionPeriod",S.periods[A]);
               S.cost = MaintenanceCost (G,S);
               Best = BacktrackingIteration (G, S, Best);
         ENDFOR
      ENDIF
   ENDIF
   RETURN Best;
END
```

```
FUNCTION IsFeasibleSolution (G: QualityGraph, T: Tuple) RETURNS BOOLEAN
   FOR EACH intermediate activity A in G DO
      If (G.getPropertyValue(A,"ActualFreshness") > G.getPropertyValue(A,"ExpectedFreshness"))
         RETURN false;
   ENDFOR
   RETURN true;
END
```

Algorithm 3.10 – Exhaustive backtracking algorithm for finding the optimal solution

In order to define pruning criteria, let's start observing an important property of the problem: the objective function (maintenance cost) is monotonic decreasing; it takes smaller values when the variables take greater values. This means that if we find a feasible solution (x_1, x_2,… x_k), all candidates tuples (y_1, y_2,… y_k) with $y_i \leq x_i$, $1 \leq i \leq k$, will have a greater or equal maintenance cost and thus, they can be pruned. This property also suggest that it is better to evaluate tuples with bigger values first; for that reason, the traversal starts with the greatest execution periods. Analogously, when the maintenance cost of a tuple is greater than that of a known solution, the branch can be pruned. Conversely, the second problem constraint is not monotonic due to the GCD function, which oscillates. This means that evaluating the constraint for a candidate tuple (x_1, x_2,… x_k) does not allow to infer its value for neighbor tuples. It makes difficult the expression of pruning criteria for this constraint.

A pseudocode of the algorithm is shown in Algorithm 3.10 (*OptimalSolution* function). It builds a candidate tuple with the greatest execution period for each activity and invokes the BacktrackingIteration function, which traverses the solution space. The base solution built by the naïve algorithm is used as current best solution. In each iteration of the *BacktrackingIteration* function, the cost of the candidate tuple is compared with the cost of the current best solution and if it is smaller, actual freshness is evaluated, invoking the ActualFreshnessPropagation algorithm (note that graph G is labeled with the execution periods of T, for allowing the evaluation of data freshness with these periods). Freshness actual values are compared with freshness expected values for each conveyance activity (IsFeasibleSolution function). If the comparison is successful, the tuple becomes the new best solution and the current branch is pruned. If not, the function iterates descending each variable in a unit of time. The execution period of the corresponding activity is updated in the graph and the maintenance cost is recalculated, then, the BacktrackingIteration function is recursively called for the new tuple.

As most branch-and-bound methods, the algorithm has combinatory complexity, and consequently it can only be executed with small size graphs. The GCD function, which is called several times per iteration during data freshness evaluation (for each data edge between intermediate activities), has also exponential order. However, as GCD arguments are bounded, the function results can be pre-calculated and stored in a matrix, which can be accessed with order 1. Next sub-sections discuss heuristics for reducing problem size.

K-Path heuristic

One of the difficulties of the problem is the synchronization of activities having several predecessors and/or successors, because improving the synchronization with one of them may degrade the synchronization with another one. However, activities that have one predecessor and one successor can be easily synchronized with their predecessors or successors without affecting other nodes. Executing these activities at different frequencies has no sense and causes the degradation of data freshness.

A first idea for reducing problem size is 0-synchronizing activities that have one predecessor and one successor, i.e. activities belonging to a K-path. K-paths are defined as follows:

> **Definition 3.17** Given a quality graph G, a *K-path* in G is a path of intermediate activities [A_1,A_2,…A_u], where the activities in the path (excepting the initial one) have only one incoming edge in G and the activities in the path (excepting the final one) have only one outgoing edge in G. □

> **Example 3.28.** In the quality graph of Figure 3.27 there are 5 K-paths, which are highlighted with shadow boxes. □

A heuristic for improving the optimal algorithm presented in previous sub-section (*K-path heuristic*), consists in assigning the same execution periods to the activities belonging to a K-path. This reduces the number of variables of the problem and therefore reduces the problem size. For example, in the quality graph of Figure 3.27, the original problem has 10 variables (x_1…x_{10}) while the heuristic problem has only 5 variables (x_1, x_2, x_4, x_6 and x_9). As the optimal algorithm has combinatorial complexity, a reduction of the problem size considerably impacts its performance. Furthermore, the bounds for the execution period of the activities in a K-path are more restrictive. The lower bound must be the greatest of processing costs (in order to can execute all activities) and the upper bound must be the least of greatest execution periods. However, it can be proved that the greatest execution period is the same for all activities in a K-path (because of the way actual and expected freshness are propagated).

The BacktrackingIteration function (see Algorithm 3.10) should be lightly modified for iterating in the K-paths instead of on intermediate activities (FOR clause) but setting execution periods of all the activities in the K-path.

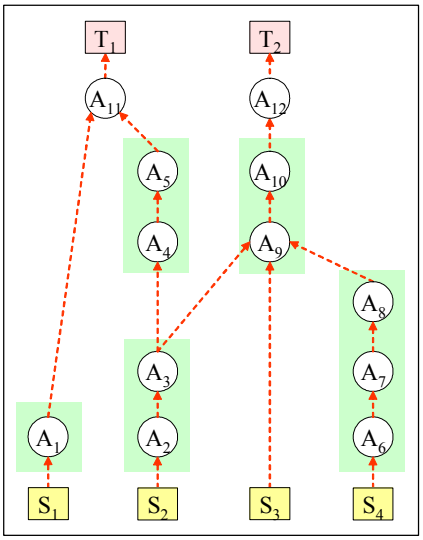

Figure 3.27 – Identification of K-paths

Random heuristic

Another idea is to build a candidate tuple with random execution periods (among solution space bounds) and then check its feasibility (using the AcutalFreshnessPropagation function). The random selection can be done for each intermediate activity or for each K-path, i.e. combined with the K-path heuristic.

The repeatedly execution of the random heuristic, a certain number of times, allows the comparison among the found solutions and the selection of the better one. Evidently, this method does not assure to find the optimal solution, however better solutions are found if we increase the number of executions. If we have a bound for the maintenance cost, we can stop when finding a good enough solution. Note that the bound may be not feasible (even for the optimal solution), so the method should also have a stop condition in the number of iterations.

If the better random solution is not good enough, it can be used as a base solution (instead of the naïve one) providing further pruning to the solution space. Furthermore, the most solutions we find, the most the solution space can be reduced. For understanding this idea, remember that the objective function is monotonic, so each time we find a feasible solution we can prune the solution space, eliminating all tuples that are smaller than the found solution.

> **Example 3.29.** Consider the two-variable solution space (x_1, x_2) shown in Figure 3.28a, for the synchronization of two intermediate activities (or two K-paths). Bounds for variables are $[a_1,b_1]$ and $[a_2,b_2]$ respectively. If we found a feasible solution (s_1,s_2), we know that all smaller tuples will not improve the objective function, so we can reduce the solution space deleting the region under (s_1,s_2), i.e. the shadow region of Figure 3.28a. Considering other feasible solutions, the solution space is yet reduced, as shown in Figure 3.28b.

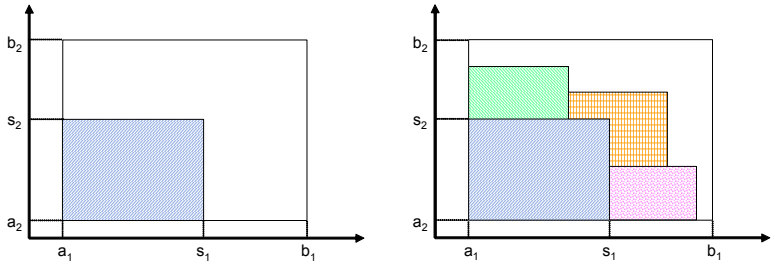

Figure 3.28 – Reduction of the solution space: (a) with one feasible solution, and (b) with several feasible solutions

The solution space can be stored in a boolean k-dimensional matrix corresponding to variables (x_1, x_2,… x_k), where a false value means that the tuple has been pruned of the solution space*. The BacktrackingIteration function can be improved checking the belonging to the matrix before iterating (instead of simply compare with processing costs; see Algorithm 3.10).

The oscillatory nature of the GCD function makes difficult the definition of local searches for optimizing a feasible solution, as in the Greedy Randomized Adaptive Search Procedure (GRASP) [Pitsoulis+2001]. However, we think that advances in operations research methods can be applied in order to find more appropriate heuristics, which is out of the scope of this thesis.

6. Conclusion

In this chapter we dealt we data freshness evaluation and enforcement topics.

We proposed a quality evaluation framework that is a first attempt to formalize the elements involved in data quality evaluation. In the framework, the DIS is modeled as a directed acyclic graph, called quality graph, which reflex the workflow structure of the DIS and contains (as labels) the DIS properties that are relevant for quality evaluation. Quality evaluation is performed by evaluation algorithms that calculate data quality traversing the quality graph.

We presented a basic algorithm for data freshness evaluation. Compared to existing evaluation proposals that only combine freshness values of source data, our algorithm takes into account two DIS properties that have impact in data freshness: the processing cost of activities and the inter-process delay among them. The algorithm can be instantiated for different application scenarios by analyzing the properties that influence the processing costs, inter-process delays and source data actual freshness in specific scenarios.

We also presented an enforcement approach for analyzing the DIS at different abstraction levels, identifying the portions that cause the non-achieved of freshness expectations. We suggested some basic improvement actions, which can be used as building-blocks for specifying macro improvement actions adapted to specific scenarios. As an application, we studied the development of an improvement strategy that follows an improvement action for a concrete application scenario. Other improvement strategies can be analyzed analogously; the quality evaluation framework and the general strategies discussed in this section (critical path, top-down analysis, actual and expected freshness propagation) may help in the analysis.

The proposal can be used at different phases of the DIS lifecycle (e.g. at design, production or maintenance phases), either for communicating data freshness to users, specifying constraints for source data or DIS development, comparing different DIS implementations accessing to alternative sources, checking the satisfaction of user freshness expectations or analyzing improvement actions for enforcing data freshness. Chapter 5 presents some applications that illustrate some of these usages.

Although we have shown that out approach can be used for DIS maintenance and evolution, we don't treat this subject in this thesis. The work of Marotta [Marotta 2006] based in our framework, treats the problem of detecting changes in source data quality and propagating changes to the DIS. They also apply improvement actions to enforce data freshness after changes. The work of Kostadinov [Mostadinov 2006] treats the expression of user preferences and then, the changes in user quality expectations. In Chapter 6, we discuss the relationship with such works as perspectives of research in these areas.

* Specific data structures for storing sparse matrices or geo-spatial data can be used.

Chapter 4. Data Accuracy

This chapter describes our proposal for data accuracy evaluation.
We reuse the framework proposed for data freshness evaluation and we propose
a data accuracy evaluation algorithm
that takes into account the distribution of inaccuracies in source relations.
We partition the query result according to data accuracy,
labeling each portion with accuracy values and we discuss how such values can be used
for enforcing data accuracy.

1. Introduction

The needs of having precise measures of data accuracy become increasingly critical in several fields. Examples are numerous:

- *Information Retrieval*: There may be a great number of web sources providing data to answer a user query and generally a big portion of retrieved data is not relevant for users because of its lacks of accuracy (e.g. hotel lists with incorrect telephone numbers or imprecise prices). The analysis of data accuracy is useful for making a pre-filtering of data or sorting data according to their accuracy.
- *Decision making*: When decision making is based on data extracted from autonomous data sources, external to the organization, a fine knowledge of data quality is necessary in order to associate relative importance to data. In this context, data accuracy should be informed to end-users, as an additional attribute qualifying data. Further strategies, as filtering inaccurate data can also be carried out.
- *Scientific experiments*: Research experiments, especially in the field of life sciences, produce large amounts of data, which are published in databanks and journals. Searches of related experiments are frequently carried out in order to cross results and abstract similar behaviors. Comparison is not trivial and it is worsen by the existence of relatively imprecise data. In this context, the analysis of data accuracy may help reducing the search space in order to retrieve the most accurate experiments.
- *Web-services integration*: When searching for a compatible service among a library of offered services, the selection is usually driven by criteria as response time or service availability. But when the service also provides data, data accuracy may play a critical role, for example, if the service provides yellow pages information.
- *Customer relationship management*: Managing inaccurate data (e.g. wrong customer addresses) may become very expensive, so knowing data accuracy becomes crucial for taking decisions. Furthermore, as many data qualifying customers are obtained from external sources (e.g. address catalogs, yellow pages, census data) with varied quality, data accuracy may be an important factor when choosing among data providers.

All these scenarios motivate the need of data accuracy evaluation methods capable of adapting to different user expectations and different perceptions of data accuracy. As argued in Chapter 2, data accuracy represents a family of quality factors. We recall the three accuracy factors that have been proposed in the literature (see Sub-section 3.1 of Chapter 2 for further details):

- *Semantic correctness* describes how well data represent states of the real-world. It captures the gap (or the semantic distance) between data represented in the system and real-world data.
- *Syntactic correctness* expresses the degree to which data is free of syntactic errors such as misspellings and format discordances. It captures the gap (or syntactic distance) between data representation in the system and expected data representation.
- *Precision* concerns the level of detail of data representation. It captures the gap between the level of detail of data in the system and its expected level of detail.

We consider all these accuracy factors. We use the term *data accuracy* when the discussion concerns all the factors and we refer to *data semantic correctness, data syntactic correctness* and *data precision* only when specific discussion is necessary. Concerning accuracy metrics, three types of metrics where described in Sub-section 3.2 of Chapter 2 for any of the accuracy factors:

- *Boolean metric*: It is a Boolean value (1=true, 0=false) that indicates if a data item is accurate or not.
- *Degree metric*: It is a degree that captures the impression or confidence of how accurate is data, commonly represented in the [0-1] range.
- *Value-deviation metric*: It is a numeric value that captures the distance between a system data item and a reference one, generally normalized to the [0-1] range.

We consider all these types of metrics. When an accuracy value must be synthesized from a set of accuracy values, (e.g. for calculating the accuracy of a source relation from the accuracy of individual cells[*]) we calculate an average of the values. Note that when values are Boolean, the average coincides with a ratio (number of accurate data items in the set divided by the total number of data items in the set). See Sub-section 3.2 of Chapter 2 for further details on aggregation functions.

In this chapter we deal with data accuracy evaluation in data integration systems (DISs). We consider a relational context; specifically, we deal with user queries consisting in selections, projections and joins over a set of source relations. We address the problem of evaluating the accuracy of the data conveyed to users in response to their queries and deciding whether users' accuracy expectations can be achieved or not.

We propose partitioning query result in areas (e.g. sets of tuples) having homogeneous accuracy and labeling such areas with their accuracy values. This allows user applications to retrieve only the most accurate data (retrieving the area with the highest accuracy), to filter data not satisfying an accuracy threshold (excluding areas having lower accuracy) or to sort data according to their accuracy (sorting areas by their accuracy). Furthermore, user applications can display first the most accurate area, and if the user wants to see more data (e.g. the result is not complete enough), they can display the following area and so on. This represents a value-added to the conveyed data.

In order to evaluate the accuracy of the data delivered to users and partition query result, we should consider how inaccuracies are distributed in source relations and how they are combined to produce query results. To this end, we partition source relations in areas having homogeneous accuracy. As argued in [Motro+1998] information sources are rarely of uniform quality, so a unique accuracy value for the whole relation may be a very crude estimation of the accuracy of specific data. Areas are defined as views (selections and projections) over the source relations. In other words, a partition constitutes a set of virtual relations defined by the predicates that characterize the partition.

We reuse the quality evaluation framework proposed for data freshness, modeling the DIS as a quality graph and reducing accuracy evaluation to a problem of value aggregation and propagation through a graph. We present an accuracy evaluation algorithm that takes into account the partitions of source relations and propagates them to query result. We focus on a priori evaluation, i.e. estimating data accuracy before executing user queries. Evaluation results can be used for comparing several query plans in order to choose the one with highest accuracy or combining the K-top plans. Evaluation results can be also used at design time (e.g. for deciding which sources to include in the DIS) and at monitoring time (e.g. for estimating accuracy of test queries). See Section 3.7.1 of Chapter 3 for a description of these usages.

Finally, we discuss the topic of accuracy improvement. We propose to use the partition of query results in order to select the areas that have the best accuracy. Note that we do not select whole relations but the portions that have the best accuracy, which differentiates our approach from the existing source selection approaches.

The following sections describe the approach: Section 2 motivates data accuracy propagation and presents an overview of our approach. Section 3 presents the background knowledge required in this chapter, especially the algorithms that are directly used for accuracy propagation. Section 4 formalizes the evaluation approach and Section 5 suggests some improvement actions. We conclude, in Section 6, by drawing the lessons learned from our experiments.

[*] The term *cell* refers to an attribute of a tuple.

2. Intuitive approach

In this section we present the intuition of our data accuracy evaluation approach. We consider a DIS in a relational context, providing a global schema that can be queried by users and accessing to a set of source relations that contain data for answering user queries. Our objective is to answer user queries using source data, sorting data in areas (sets of tuples) that have homogeneous accuracy and informing users of the accuracy of such areas. For example, if a user query asks for students' data, we can answer saying '*Student data with 5% of inaccuracies is: ..., student data with 10% of inaccuracies is: ... and so on*'. We describe an *a priori* evaluation strategy, i.e. data accuracy of query results are estimated before executing queries based on estimations of accuracy of source data and on the way of combining it. In this way, only the data that satisfy user's accuracy expectations will be extracted from sources and conveyed in response to the query.

The following example will be used along the chapter for illustrating our evaluation approach. We exemplify the measurement of semantic accuracy with the Boolean metric, but the same discussion can be done for the other accuracy factors and metrics.

Example 4.1. Consider the global schema of a DIS containing two relations:

- S (stid, name, nationality, address, city, telephone, interview, test)
- M (stid, year, mark)

which provide information about students and their annual average marks respectively. Attributes describing students are: stid (the student identification number), name, nationality, address, city, telephone, interview (initial level determined by interviews; taking values 'high', 'medium' or 'low') and test (initial test result; taking values between 0 and 1). Attributes describing marks are: stid (the student identification number), year and mark (taking values in the 0-10 range, where 10 is the maximal mark). The keys of the relations are {*stid*} and {*stid, year*} respectively.

Consider two sources providing information about students and marks:

Source$_1$*:*

- Students (stid, name, interview, test, address, telephone) // students living at Montevideo.
- Marks (stid, year, mark) // marks of those students

Source$_2$*:*

- Classing (stid, name, nationality, interview, test) // students having a test punctuation superior to 0.8.

Along the chapter, for illustrating some techniques, we will refer to the instances of the *Students* and *Marks* relations of *Source*$_1$, which are shown Table 4.1 and Table 4.2 respectively. Accuracy values are illustrated coloring the inaccurate cells (Boolean metric). The accuracy values of the *Classing* relation are illustrated in Figure 4.2a also coloring inaccurate cells. Aggregated accuracy values for those relations (obtained as an average of accuracy of cells) are 0.60 (28/60), 0.80 (36/45) and 0.82 (102/125) respectively.

Consider the following queries accessing to *S* and *M*:

- *UserQuery*$_1$ asks for students (stid, name, nationality, interview and test) that obtained 'high' level during interview
- *UserQuery*$_2$ asks for names and marks of students in year 2005 (keys are also projected)

Figure 4.1 illustrates the queries expressed in relational algebra.

Table 4.3 shows a possible answer to *UserQuery*$_2$, obtained extracting data from the *Students* and *Marks* relations by performing the query $Q_2=\pi_{stid,year,name,mark}(Students \bowtie_{stid} \sigma_{year='2005'}(Marks))$. Colored cells correspond to inaccuracies. The accuracy of query result is 0.70 (28/40). Analogously, Figure 4.2d shows a possible answer to *UserQuery*$_1$, obtained extracting data from the *Classing* relation by performing the query $Q_1=\sigma_{interview='high'}(Classing)$. The accuracy of query result is 0.94 (66/70). □

stid	name	interview	test	address	telephone
21	María Roca	low	1.0	Carrasco	6001104
22	Juan Pérez	medium	.5	Coloniaa 1280/403	9023365
43	Emilio Gutiérrez	high	.8	Irigoitía 384	3364244
56	Gabriel García	low	.5	Propios 2145/101	
57	Laura Torres	medium	.7	Maldonado & Yaro	099628734
58	Raúl González	high	9	Rbla Rca Chile 1280/1102	4112533
101	Carlos Schnider	high	.9701	Copacabana 1210	094432528
102	Miriam Revoir	medium	.7945		9001029
103	A. Benedetti	low	.9146	Charrúa 1284/1	7091232
104	Luis López	high	.8220	Sixtina s/n	

Table 4.1 – *Students* relation

stid	year	mark
21	2005	7
43	2005	10
43	1004	8
56	2004	9
57	2004	3
57	2005	4
58	2005	6
101	2004	9
101	2005	10
102	2004	7
102	2005	10
103	2004	8
103	2005	6
104	2004	10
104	2005	9

Table 4.2 – *Marks* relation

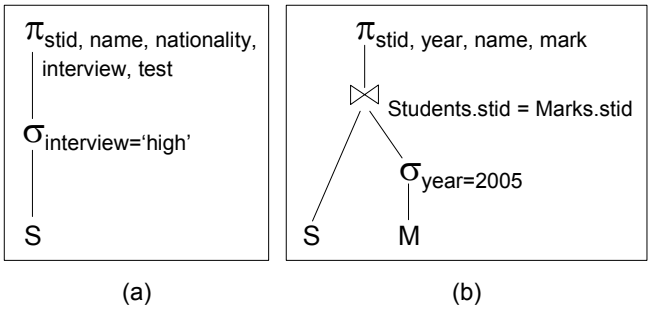

Figure 4.1 – User queries: (a) *UserQuery*$_1$, and (b) *UserQuery*$_2$

stid	year	name	mark
21	2005	María Roca	7
43	2005	Emilio Gutiérrez	10
57	2005	Laura Torres	4
58	2005	Raúl González	6
101	2005	Carlos Schnider	10
102	2005	Miriam Revoir	10
103	2005	A. Benedetti	6
104	2005	Luis Lopez	9

Table 4.3 – Answer to *UserQuery*$_2$ extracting data from the *Students* and *Marks* relations

Our approach for accuracy evaluation consists in two main phases: (i) partitioning source relations according to accuracy homogeneity in order to represent the distribution of inaccuracies, and (ii) for each user query, partitioning query result based on the partition of source relations.

We partition each source relation in areas (sets of tuples) that are highly homogeneous with respect to their accuracy*. Homogeneity means that any sub-area of a highly homogeneous area would maintain roughly the same accuracy as the initial area. Homogeneity does not mean that all cells in an area have the same accuracy value, but that they can be considered as having the same accuracy value. Areas are defined as views (selections) over the relations. In other words, areas constitute virtual relations defined by the predicates that characterize the partition (the selection predicates) and consequently they can be treated as any relation.

The accuracy of an area can be calculated as the average of the accuracy of its cells (or a sample of cells). The accuracy of cells can be measured using any of the techniques discussed in Chapter 2. When some knowledge about source data is available (e.g. 10% of data about foreign customers is inaccurate), it can be used for estimating accuracy. Such knowledge can be provided by source or domain experts or derived from users' feedback on previously queried data. Note that as areas have homogeneous accuracy, the accuracy of a source relation can also be estimated from the accuracy of its areas, as a weighted sum, where weights are the number of cells in the areas.

* In Section 4, we provide a more complete definition of partitioning, in which areas can be partitioned in sub-areas (sets of attributes) in order to better represent the distribution of inaccuracies. We use a simpler intuition here in order to clearly motivate the proposal.

Example 4.2. Consider the *Classing* relation illustrated in Figure 4.2a (colored cells correspond to inaccuracies). The relation is partitioned in three areas C_1, C_2 and C_3, i.e. *Classing* = $C_1 \cup C_2 \cup C_3$, with 10, 8 and 7 tuples (i.e. 50, 40 and 35 cells) respectively. Accuracy values are aggregated for areas, obtaining the values 0.98 for C_1, 0.90 for C_2 and 0.50 for C_3. Figure 4.2b shows the partition, coloring areas with different colors. The accuracy of the relation *Classing* (calculated in previous example as 0.82) can also be calculated from the accuracy of its areas, obtaining (50*0.98 + 40*0.90 + 35*0.50) / (50+40+35) = 0.82. □

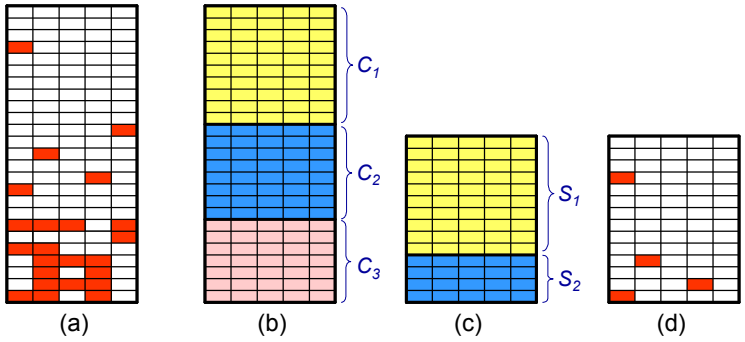

Figure 4.2 – (a) Distribution of inaccuracies in the *Classing* relation (colored cells correspond to inaccuracies), (b) partition of the relation, (c) partition of the answer to *UserQuery*$_1$ extracting data from the *Classing* relation, (d) distribution of inaccuracies in the query answer

In order to build a partition of query result, note that as areas are virtual relations they can be considered as source relations providing data for answering the user query. In other words, given a source relation R partitioned in areas $R_1,\ldots R_n$ and an extraction query Q over R, we can define a set of queries $Q_1 \ldots Q_n$ with the same semantics as Q, for extracting data from areas $R_1 \ldots R_n$, respectively. We call them sub-queries because they extract a portion of the data returned by Q. For example, if $Q=\sigma_P(R)$, we can define $Q_i=\sigma_P(R_i)$, $1 \le i \le n$. Furthermore, under certain hypotheses (a set of conditions that will be presented in Sub-section 3.1.4) the union of sub-queries is equivalent to the original query, i.e. $Q \equiv Q_1 \cup \ldots \cup Q_n$.

Each sub-query over an area of a source relation returns a set of tuples (eventually empty) that are contained in the result of the original query over the source relation. In other words, the query result is partitioned in areas, each one corresponding to the result of a sub-query. Furthermore, as areas of the source relation have homogeneous accuracy, a subset of their tuples maintains the accuracy, and therefore, the areas of the query result also have homogeneous accuracy.

Example 4.3. Continuing previous example, consider the query $Q_1=\sigma_{\text{interview='high'}}(Classing)$ extracting data from the *Classing* relation in order to solve *UserQuery*$_1$. The sub-queries S_1, S_2 and S_3 over C_1, C_2 and C_3, defined as $S_i=\sigma_{\text{interview='high'}}(C_i)$, $1 \le i \le 3$, partition the result of Q_1, as shown in Figure 4.2c (note that Q_3 does not return any tuple).

The accuracy of S_1, S_2 and S_3 is estimated as the accuracy of C_1, C_2 and C_3, i.e. 0.98, 0.90 and 0.50 respectively, because of the accuracy homogeneity of C_1, C_2 and C_3. □

When several source relations provide the same type of data, the areas of those relations can be combined together. For example, the areas of the *Students* and *Classing* relations can be combined for answering *UserQuery*$_1$. Furthermore, if the user query joins several relations, sub-queries may join areas that partition several source relations.

Remember that user queries are expressed in terms of the global schema so we need to reformulate (rewrite) them over areas. The problem of rewrite user queries in terms of source relations has been largely studied (it is recalled in Sub-section 3.2). We propose to use query rewriting algorithms for combining areas of source relations instead of combining the whole relations.

Once sub-queries have been generated and having an accuracy estimation for them, we can aggregate an accuracy value for the whole result (as we have done for source relations) by performing a weighted sum of the areas of the result, where weights are the number of cells in each area. To this end, we need to estimate the number of projected attributes and the number of selected tuples in each sub-query. While the former is simple to

obtain (it is structural), the latter needs the estimation of the selectivity of the query. Note that we do not need to know *which* tuples will be returned by each sub-query but *how many* tuples will be returned.

> **Example 4.4.** Continuing previous example, if we estimate that S_1 returns all tuples of C_1 (50 cells), S_2 returns a half of tuples of C_2 (20 cells) and S_3 returns no tuples, the accuracy of Q_1 is calculated as (50*0.98 + 20*0.90 + 0*0.50) / (50+20+0) = 0.96. □

As areas are not perfectly homogeneous and their accuracy may be an estimation, the accuracy measure obtained for query result is an approximation, not necessarily the exact value that can be computed knowing the precise query result (i.e. with *a posteriori* evaluation). In previous example, we obtained the value 0.96 instead of the exact value 0.94 = 66/70 obtained from the distribution of inaccuracies shown in Figure 4.2d. But note that the most homogeneous are the areas and the most precise are their accuracy measures, the most precise is the accuracy estimation for the areas of query result. In addition, if the estimation of the size of sub-queries results is good enough, the final accuracy aggregation should be near to the exact accuracy value. Then, the precision of the accuracy estimation relies on the methods used for estimating accuracy of source relations, partitioning them and estimating selectivity of sub-queries.

In summary, our proposal for accuracy evaluation consists in three steps:

1. *Partitioning source relations according to accuracy homogeneity*: This step consists in estimating data accuracy of a sample of each source relation and using accuracy estimations for partitioning the relations, conforming areas of homogeneous accuracy. To this end, we adapt the partitioning algorithm proposed in [Rakov 1998], which is presented in Sub-section 3.1.3. This step is executed once at DIS construction or periodically (but it is not executed for each user query).
2. *Rewriting user queries in terms of partitions*: Each user query is rewritten in terms of the areas that partition source relations. Specifically, a query is expressed as the union of a set of sub-queries, each one accessing to only one area of each source relation. We utilize the Bucket algorithm [Levy+1996] for generating the sub-queries, which is recalled in Sub-section 3.2. The rewriting algorithm can also detect some cases of non-contributive sub-queries, i.e. sub-queries returning no tuples.
3. *Estimating data accuracy of query results*: In this step, we estimate an accuracy value for each sub-query (area in the result) from the accuracy of areas of input relations. This estimation depends on the sub-query operations (selection, projection, join). In addition, the query result is expressed as the union of the sub-queries and an accuracy value is aggregated for query result, as the weighted sum of their accuracy, where weights are the numbers of cells returned by sub-queries. In order to obtain those weights, we estimate the selectivity of operations. Some techniques for selectivity estimation are summarized in Sub-section 3.3.

The quality evaluation framework proposed for data freshness is extended for the evaluation of data accuracy. In this context, quality graphs represent user queries (rewritten in terms of partitions) and are labeled with query properties (such as query selectivity) necessaries for accuracy evaluation. The framework is recalled in Sub-section 3.4.

Next section describes the techniques and algorithms that are used in our evaluation approach, and then, Section 4 formalizes the approach.

3. Background

In this section we review the concepts that are used throughout this chapter. We firstly discuss some existing approaches for accuracy evaluation that we adapt to our framework, in particular, a partitioning algorithm, and we recall some properties about fragmentation in the relational model. We also recall the query rewriting principle and we explain the Bucket algorithm. Then, we comment some techniques for selectivity estimation. We end the section recalling the quality evaluation framework presented in Chapter 3.

3.1. Some related approaches for accuracy evaluation

Our approach for accuracy evaluation is based on evaluation techniques proposed in two works: (i) an a priori evaluation approach, which assumes uniform distribution of errors [Naumann+1999], and (ii) an approach for a posteriori evaluation, which partitions source relations according to accuracy homogeneity [Rakov 1998]. The following sub-sections describe these approaches.

3.1.1. Techniques for a priori evaluation

A methodology for propagating quality values (including data accuracy) along query operators was proposed in [Naumann+1999]. The approach consists in estimating the accuracy of query results based exclusively in the accuracy of source data. Accuracy of source relations (set granularity) is estimated as the ratio of syntactic correctness (the percentage of cells without errors).

Queries are JSP (join, selection, projection) queries. Authors consider that errors are uniformly distributed in the source relations, so no matter which attributes are projected or which tuples are selected, the accuracy of the source relations is preserved. For the join operation, the accuracy of the joined data is calculated as the product of the accuracy of both input relations. The following example illustrates the approach:

> **Example 4.5.** Consider the query $Q_1 = \sigma_{\text{interview='high'}}(Classing)$ introduced in Example 4.1. Selections preserve the accuracy value because of the hypothesis of uniform distribution of errors, so, accuracy of query result is estimated as the accuracy of *Classing*, i.e. 0.82. However, the accuracy of query result is 0.94 (obtained from the distribution of inaccuracies illustrated in Figure 4.2). □

The weak point of the Naumann's approach is the strong hypothesis on uniform distribution of inaccuracies which is rarely applicable to real data. In most cases, query operations do not preserve accuracy values (e.g. query Q of previous example) and consequently, we do not obtain precise estimations of the accuracy of query results. The main problem is that we do not know where inaccuracies are concentrated (some attributes, some sets of tuples). Additional information describing relation instances is necessary to obtain more precise results. We propose to estimate the distribution of inaccuracies and thus partition source relations.

In next sub-section we discuss an approach for partitioning source relations according to data accuracy, which has been proposed for a posteriori evaluation but can be adapted for a priori evaluation.

3.1.2. Techniques for a posteriori evaluation

An algorithm for partitioning source relations in areas that are highly homogeneous with respect to accuracy was proposed in [Rakov 1998] [Motro+1998]. Areas are defined with views, which may involve selections (with conjunctive conditions) and projections. The accuracy measurement is performed by taking a sample* of each source relation and measuring accuracy of the cells of the sample. Accuracy values for areas (set granularity) are ratios of semantic correctness (percentage of cells that correspond to real-world items). The accuracy values are used for partitioning the sample, using an automatic partitioning algorithm (Algorithm 4.1) that tests different partitioning criteria. Then, the same partition is applied to the whole relation.

User queries are conjunctive queries; query operators are selection, projection and Cartesian product. The relational algebra is extended for operating with the partitions, i.e. operators take as input the relations and their partitions and return a relation and its partition. For example, the partition of a selection or projection is computed intersecting operation result with the partition of the input relation (intersecting projected attributes and selection conditions); accuracy is preserved because of accuracy homogeneity. At the end of the query, a unique accuracy value is calculated for the query result as a weighted sum of the accuracy of the areas (weights are the number of cells in each area). The following example illustrates the approach:

> **Example 4.6.** Consider the query $Q_1 = \sigma_{\text{interview='high'}}(Classing)$ presented in Example 4.1 and the partition of the *Classing* relation in the areas C_1, C_2 and C_3 shown in Figure 4.2b. Areas are defined by certain selection predicates P1, P2 and P3, i.e. $C_i = \sigma_{Pi}(R)$, $1 \le i \le 3$. Accuracy of areas is 0.98 for C_1, 0.90 for C_2 and 0.50 for C_3.
>
> The partition of Q_1 results from intersecting the partition of the *Classing* relation with the selection conditions, i.e. $S_i = \sigma_{Pi \wedge \text{interview='high'}}(Classing)$, $1 \le i \le 3$. The extension of S_1, S_2 and S_3 is obtained computing the query, in other words, we know which tuples of each area are selected. Consequently, we know the number of cells of each area that are selected (50, 20 and 0 respectively, as shown in Figure 4.2c). The numbers of selected cells are used as weights for aggregating the accuracy of the result from the accuracy of areas, obtaining: (50*0.98 + 20*0.90 + 0*0.50) / (50+20+0) = 0.96. □

* Typical size for the sample is 10% of the source relation [Rakov 1998].

The Rakov's approach is based in the knowledge of query result. Specifically, his extended algebra intersects partitions with query results. The difference with our approach is that we do not know which tuples are returned by the query (because we perform a priori evaluation) but we estimate how many tuples may be returned by the query. However, our approach is inspired by the notion of partition of the Rakov's approach. We reuse the partitioning algorithm (which is detailed in Sub-section 3.1.3) and the principle for calculating accuracy of query results as a weighted sum of accuracy of areas.

Another a posteriori approach is proposed in [Laboisse 2005]. They propose measuring and storing accuracy values of source relation cells and storing accuracy values as additional attributes of source relations (called *quality attributes*). Then, when executing user queries quality attributes are also selected, so the accuracy values can be aggregated, obtaining a measure of the accuracy of query result. As this method obtains the precise accuracy value or query result, it will be used for comparing evaluation results.

3.1.3. Partitioning algorithm

This sub-section briefly describes the algorithm for partitioning a relation proposed in [Rakov 1998], whose pseudocode is shown in Algorithm 4.1. It is based on regression and classification trees. It proceeds iteratively, starting at the relation and splitting it in two areas (either horizontally or vertically but not both), then repeating the procedure for each area and so on. At each step, it finds the split that gives maximum gain in homogeneity. The splitting of an area stops when it can provide only marginal improvement in homogeneity (a threshold t is used as stop condition). This indicates that this area has a fairly homogeneous distribution of inaccuracies.

Homogeneity is approximated by Gini indexes [Breiman+1984]. The Gini index $G(v)$ is calculated as $G(v)=2p(1-p)$, where p denotes the proportion of correct cells in a view v. The stop condition calculates the split's reduction of Gini index: $\Delta G = G(v) - \alpha_1 G(v_1) - \alpha_2 G(v_2)$, where $\{v_1, v_2\}$ is a split of v and $\alpha_i = |v_i|/|v|$, i=1,2.

As considering all possible splits of a relation is extremely expensive*, Rakov proposes heuristics for reducing the number of splits considered. Recall that there are two sorts of attributes: ordered and categorical. For horizontal splits, if an attribute is ordered and has k distinct values ($a_1 \leq \ldots \leq a_k$), they consider the k-1 binary conditions $x \leq a_i$ as possible splits. If an attribute is categorical and has l distinct values, they order these values according to the number of inaccurate cells in the tuples having these values, and they treat them as ordered attributes. For vertical splits, if the number of attributes is small, all possible splits can be considered, but if it is large (say, n>20) the same strategy used for categorical attributes can be applied.

```
FUNCTION Partitioning (t: THRESHOLD, R: Relation, G: QualityGraph) RETURNS LIST OF Views
   LIST OF Views V = {};
   QUEUE Q = {R};
   WHILE Q is not empty DO
      Get the next element N of Q;
      Consider all possible splits of N and choose the maximal split s of N;
      IF ΔG(s) * NumberCells(N) ≥ t
         Split N and put the two resulting views in Q;
      ELSE
         Add N to V;
   ENDWHILE;
   RETURN V;
END
```

Algorithm 4.1 – Algorithm for finding a partition of a relation (taken from [Rakov 1998])

This notion of partition is similar to the notion of fragmentation largely studied in distributed databases (see for example [Ozsu+1991]). Next sub-section recalls properties of well-formed fragmentations.

* There are 2^{m-1}-1 possible horizontal splits and 2^{n-1}-1 possible vertical splits, being m the number of tuples and n the number of attributes of the relation.

3.1.4. Correctness of fragmentations

Özsu and Valduriez [Ozsu+1991] enunciated three correctness rules that a fragmentation should verify in order to assure database consistency. The rules are:

- *Completeness*: If a relation instance R is decomposed into fragments $R_1 \ldots R_n$, each cell that can be found in R can also be found in one or more R_i. This property ensures that the data in a global relation is mapped into fragments without any loss. In the case of horizontal fragmentation the "item" typically refers to a tuple while in the case of vertical fragmentation it refers to an attribute.
- *Reconstruction*: If a relation R is decomposed into fragments $R_1 \ldots R_n$, it should be possible to define a relational operator ∇ such that $R = \nabla R_i$, $i = 1..n$. This property ensures that constraints defined on the data in the form of dependencies are preserved. In the case of horizontal fragmentation ∇ is typically the union operation while in the case of vertical fragmentation it is the join operation.
- *Disjointness*: If a relation R is horizontally decomposed into fragments $R_1 \ldots R_n$, and cell d_i is in R_j, it is not in any other fragment R_k, $k \neq j$. This criterion ensures that the horizontal fragments are disjoint. If a relation R is vertically decomposed, its primary key attributes are typically repeated in all its fragments. Therefore, in case of vertical fragmentation, disjointness is defined only on the non-primary key attributes of a relation.

Our proposal partitions (fragment) source relations horizontally and vertically. Horizontal partitions must verify the previous rules, i.e. a tuple must belong to one and only one fragment, allowing reconstructing the original relation using the union operator. Vertical partitions must also verify the rules, i.e. a non-key attribute must belong to one and only one fragment. Key attributes must belong to all fragments, allowing reconstructing the original relation using the join operator.

Next sub-section describe another technique reused in this work: query rewriting.

3.2. Query rewriting

In this sub-section we recall the concept of query rewriting and we describe a rewriting algorithm. *Query rewriting* consists in reformulating a user query (expressed in terms of the global schema) into a (possibly) equivalent expression, called *rewriting*, that refers only to the source structures [Calvanese+2001].

In the *local-as-view* (LAV) approach, the global schema is specified independently of the data sources and the mappings between them are established by defining every source relation as a view over the global schema. Expressing sources relations as views over the global schema, the query rewriting problem is similar to the problem of answering queries using views.

Given a query Q over relations $E_1, \ldots E_n$ (the global schema) and a set of views $V=\{V_1, \ldots V_m\}$ (the source relations) over $E_1, \ldots E_n$, a query Q_r is a rewriting of Q if: (i) it is contained in Q and, (ii) it refers only to $V_1, \ldots V_m$.

In most works, queries are conjunctive select-project-join queries and are expressed in Datalog-like notation. A query Q has the form:

$$Q(X) \leftarrow R_1(Z_1) \wedge \ldots \wedge R_n(Z_n) \wedge C_Q$$

where:

- $R_1 \ldots R_2$ are relations; Z_i is a set of variables representing the attributes of R_i*
- C_Q is a conjunction of predicates of the form $u \theta v$ where $\theta \in \{=,<,>,\leq,\geq\}$ and $u, v \in \bigcup_{1 \leq i \leq n} Z_i$
- $X \subseteq \bigcup_{1 \leq i \leq n} Z_i$ is a set of variables representing the attributes projected by Q

Given two queries Q_1 and Q_2, we say that Q_1 is contained in Q_2, denoted $Q_1 \subseteq Q_2$, if for all databases the set of tuples returned by Q_1 is included in the set of tuples returned by Q_2. Query containment has been studied in various works; see for example [Chandra+1977] [Chekuri+1997] [Klug 1988] [van der Meyden 1992].

Several query rewriting algorithms have been proposed; a survey of methods is presented in [Calvanese+2001]. We briefly describe the *Bucket* algorithm [Levy+1996], which will be used later in this chapter.

* Some works allows representing constants in the domain of the attributes. We prefer, for ease of understanding, to represent constants with additional equality conditions. But the two styles are compatible and isomorphic.

The Bucket algorithm aims at computing all the rewritings that are contained in (and not necessarily equivalent to) the original query, by pruning the space of candidate rewritings. It proceeds in two steps:

1. For each atom *g* in Q, create a bucket that contains the source relations that are contributive for *g*, i.e. the source relations from which tuples of *g* can be obtained.
2. Build candidate rewritings (conjunctive queries obtained by combining one source relation from each bucket) and keep only rewritings that are contained in Q.

The first step, whose running-time is polynomial in the number of sources, considerably reduces the number of possibilities considered in the second step. Although containment is intractable in general, its intractability is in the size of the query (which tends to be small) and only occur when queries have multiple occurrences of the same relations; consequently, the complexity of containment is not a problem in practice [Levy+1996].

The following sub-section reviews techniques for selectivity estimation.

3.3. Selectivity estimation

Selectivity estimation techniques are largely used in the field of query optimization; they use statistical information about the data that is stored in the database system to provide estimates to the query optimizer. Histograms are the most common statistical information used in commercial database systems. In this sub-section we show how they are currently used to estimate the selectivity of complex queries.

A histogram on attribute x consists of a set of buckets. Each bucket b_i represents a sub-range r_i of x's domain, and has associated two values: (i) the frequency f_i, representing the number of tuples t verifying $t.x \in r_i$, and (ii) the distinct value dv_i, representing the number of distinct values of $t.x$ among all the tuples t satisfying $t.x \in r_i$. The main assumption (*uniform spread assumption*) is that each bucket b_i is composed of dv_i equidistant groups of $\delta_i = f_i/dv_i$ tuples each (δ_i is called the density of the bucket) [Bruno+2002]. This assumption suggests a natural interpolation-based procedure to estimate the selectivity of range and join predicates.

To estimate the cardinality of queries with range predicates using a histogram on the range attribute, the frequencies of histogram buckets that are completely or partially covered by the predicate are added (prorating partially covered buckets). Selectivity is obtained dividing query cardinality by relation cardinality.

> **Example 4.7.** Consider the query $Q_3=\sigma_{R.a<15}(R)$. Using the histogram on attribute *R.a* shown in Figure 4.3c the cardinality of Q_3 is estimated adding the frequency of bucket b_1 (which is completely contained) and half of the frequency of bucket b_2 (which is contained at 50% assuming uniform spread), i.e. 60 + 20*0.50 = 70 tuples; selectivity is 70/100=0.70. □

When queries have multiple range predicates, assuming attribute independence (*independence assumption*) selectivity is estimated as the product of the selectivity of each predicate. For example, given the query $\sigma_{P1 \wedge P2}(R)$ where s_i is the selectivity of predicate P_i, $1 \leq i \leq 2$, the selectivity of $P_1 \wedge P_2$ is estimated as $s_1 * s_2$. Multidimensional histograms have been also proposed for the case of non-attribute independence (see for example [Poosala+1997]).

The method to estimate the cardinality of queries with join predicates using histograms on the join attributes consists of three steps [Bruno+2002]. In the first step, both histograms are aligned so that their boundaries agree (splitting some buckets). In the second step, each pair of aligned buckets is analyzed estimating the join size. Assuming that tuples belonging to the bucket with minimal distinct values joins with some tuples of the other bucket (*containment assumption*), the density of the resulting bucket is calculated multiplying the density of input buckets, and frequencies and distinct values are recalculated. In the last step, the cardinality of the join is estimated adding frequencies of buckets. Selectivity is estimated dividing cardinality by the product of cardinalities of input relations.

> **Example 4.8.** Consider the query $Q_4= R \bowtie_{R.a=S.b} S$. Using the histograms on attributes *R.a* and *S.b* shown in Figure 4.3a and Figure 4.3b, the estimation of the cardinality of Q_4 proceeds as follows: Firstly, buckets of both histograms are aligned and frequencies and distinct values are prorated (Figure 4.3c and Figure 4.3d). Then, densities of aligned buckets are calculated multiplying input densities, e.g. $\delta_1'' = 120$ (60/2 * 20/5), and frequencies and distinct values are aggregated (Figure 4.3e), e.g. $dv_1''=2$ (minimum of dv_1 and dv_1') and $f_1'' = 240$ ($\delta_1'' * dv_1''$). Finally, join cardinality is aggregated, obtaining 440 (240+40+160) tuples. Selectivity is 0.37 = 440 / (100*120). □

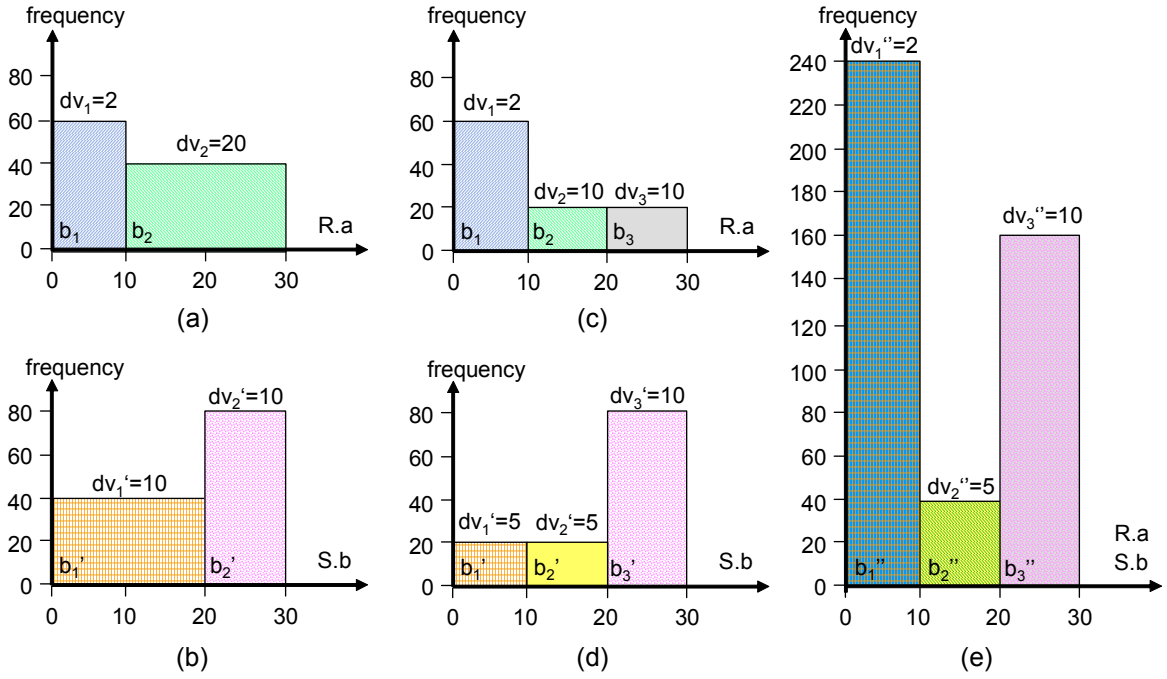

Figure 4.3 - Join selectivity estimation using histograms

When considering arbitrary SPJ queries, we face an additional challenge: cardinality estimation requires propagating statistics through predicates. In other words, we need to build histograms for intermediate results based on the histograms of the base relations.

> **Example 4.9.** Consider the query $Q_5=\sigma_{R.a<15}(R \bowtie_{R.a=S.b} S)$ and the histograms shown in Figure 4.3a and Figure 4.3b. The histogram build for the intermediate result $R \bowtie_{R.a=S.b} S$ may be used for estimating selectivity of the selection. □

As statistics of intermediate results are estimations, their propagation along several query operators may considerably lack of precision. An alternative consists in precalculating statistics of some distinguished query results and using them for calculating selectivities of intermediate results. For example, if statistics for the $R \bowtie_{R.a=S.b} S$ intermediate result are available, they can be used in the estimation of the selectivity of query Q_5. In [Acharya+1999], authors propose the use of *join synopses* (precomputed samples of a small set of distinguished joins with foreign-key predicates) for building statistics of intermediate results. A more general approach covering JSP queries is presented in [Bruno+2002]. Authors propose to build *SITs* (statistics built on attributes of the result of a query expression) and present methods for selecting the most relevant SITs to compute.

3.4. Quality evaluation framework

This sub-section briefly recall the framework for data quality evaluation presented in Section 2 of Chapter 3. The framework models the DIS processes and properties and evaluates the quality of the data conveyed to the user. The goal of the framework is twofold, firstly, helping in the identification of the DIS properties that should be taken into account for data quality evaluation, and secondly, allowing the easy development of evaluation algorithms that consider such properties.

The framework consists of: (i) a set of data sources, (ii) a set of data targets, (iii) a set of quality graphs representing several DISs processes, (iv) a set of properties describing DIS features and quality measures, and (v) a set of quality evaluation algorithms.

A 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. Each workflow activity takes data from sources or other activities and produces result data that can be used as input for other activities. Then, data traverses a path from sources to targets where it is transformed and processed according to the system logics. A *quality graph* is a

graph that has the same workflow structure as the DIS and is adorned with property values useful for quality evaluation. The nodes are of three types: (i) *activity nodes* representing the major tasks of a DIS, (ii) *source nodes* representing data sources accessed by the DIS, and (iii) *target nodes* representing data targets fed by the DIS. There are two types of edges: (i) *control edges* expressing the control flow dependencies between activities, and (ii) *data edges* representing data flow from sources to activities, from activities to targets and between activities. In the DISs considered in this chapter, control flow is induced by data flow, i.e. there is a control flow edge between two activities if and only if there is a data flow edge between them. Nodes and edges of quality graphs are adorned with property labels of the form *property = value*. Properties can be of two types: (i) *features*, indicating some characteristic of the DIS (costs, delays, policies, strategies, constraints, etc.), or (ii) *measures*, indicating a quality value corresponding to a quality factor. Figure 4.4 illustrates a quality graph.

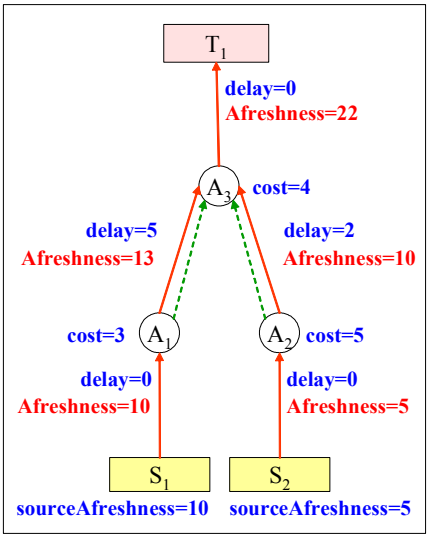

Figure 4.4 – Quality graph

The quality evaluation is performed by evaluation algorithms. As a quality graph describes the DIS integration process and its properties, it contains the input information needed by evaluation algorithms. Evaluation algorithms take as input a quality graph, calculate the quality values corresponding to a quality factor and return a quality graph with an additional property (corresponding to the evaluated quality factor). Evaluation algorithms may traverse the graph, node by node, operating with property values; this mechanism is called *quality propagation*. Concerning code, evaluation algorithms have the following signature:

```
FUNCTION AlgorithmName (G: QualityGraph) RETURNS QualityGraph
```

The implementation of evaluation algorithms may vary according to the quality factor and the concrete application scenario. The framework does not constrain the way the algorithms can be implemented.

Next section formalizes our approach for data accuracy evaluation. The techniques and algorithms described in this section are adapted and used during accuracy evaluation.

4. Formal approach

In this section we formalize our approach for accuracy evaluation. Accuracy can be measured using any of the techniques discussed in Chapter 2. Our approach is independent of the accuracy metric used but of course the accuracy measures depend on it. We consider a DIS in a relational context, i.e. it integrates data of relational sources and allows users to pose JSP (join, selection, projection) queries over a relational global schema. User queries are reformulated (rewritten) in terms of the source relations, obtaining a set of query rewritings that provide data for answering the user query.

We address the problem of estimating the accuracy of the data conveyed to users in response to a query. We organize data in areas of homogeneous accuracy in order to inform users (or user applications) about the distribution of inaccuracies. To this end, source relations are partitioned in areas having homogeneous accuracy.

Areas are virtual relations (views) defined according to the partitioning criteria (selection conditions and projection attributes). User queries are rewritten over these areas (instead of over whole relations).

We propose an *a priori* evaluation strategy, i.e. data accuracy is estimated before executing the rewriting queries, based exclusively on the accuracy of the areas of the source relations and on the operations (selection, projection, join) that define the rewriting.

The problem can be stated as follows:

Given:
- *a set of source relations partitioned in areas of homogeneous accuracy, and*
- *a user query over the global schema,*

Obtain:
- *query rewritings (over areas of source relations) that answer the query, and*
- *estimations of the accuracy of their data*

Our proposal for accuracy evaluation consists in three steps:

1. *Partitioning source relations according to accuracy homogeneity*: This step consists in estimating data accuracy of a sample of each source relation and using accuracy estimations for partitioning the relations. This step is executed in a preparation phase, at DIS construction or periodically, but separated from the query evaluation phase. It is described in Sub-section 4.1.
2. *Rewriting user queries in terms of partitions*: Each user query is rewritten in terms of the areas that partition source relations. This step consists in generating a set of query rewritings over these areas. The result to the user query consists of the union of the generated rewritings. This step is explained in Sub-section 4.2.
3. *Estimating data accuracy of query results*: In this step, we estimate the accuracy of the data returned by each rewriting, based on the accuracy of areas of source relations, and we aggregate an accuracy value for query result. The estimation is described in Sub-sections 4.3 and an accuracy evaluation algorithm is proposed in Sub-section 4.4.

In order to enforce data accuracy, a simple improvement action consists in discarding rewritings (or directly areas of source relations) that have low accuracy. Note that areas can be discarded in an early phase (during the generation of the rewritings) so our proposal correspond to a *selective rewriting* strategy. This point is discussed in Sub-section 5.

The following sub-sections describe each step.

4.1. Partitioning of source relations according to accuracy homogeneity

Inspired by the partitions proposed by Rakov [Rakov 1998], we partition each source relation according to data accuracy. The idea behind partitioning is to manipulate portions of the relation that have homogeneous accuracy, i.e. if we partition again, in any manner, the accuracy estimation will not considerably change. In this sub-section we define partitions and we discuss how a convenient partition can be obtained.

For easy manipulation of partitions (which will be explained in next sub-section), we firstly partition the relation in *areas* according to selection predicates (horizontal partition) and then, we partition each area in *sub-areas* according to sub-sets of attributes (vertical partition). Partitions must verify the three correctness rules discussed in Sub-section 3.1.2, i.e. a tuple must belong to one and only one area, a non-key attribute must belong to one and only one sub-area and key attributes must belong to all sub-areas.

In order to treat all attributes (key and non-key attributes) in the same manner, i.e. projecting them in only one sub-area, we duplicate key attributes, generating a new key for the relation (that composed of the new attributes). This strategy was introduced by Rakov [Rakov 1998] with the name of *key expansion*.

Definition 4.1 (key expansion). Given a relation $R(A_1,\ldots A_m,\ldots A_n)$, $m \leq n$, where $A_1,\ldots A_m$ constitute a key of R, a *key expansion* of R, denoted $\overline{R}$, is a relation obtained replicating the key attributes: $\overline{R}(K_1,\ldots K_m,A_1,\ldots A_m,\ldots A_n)$. The replicated attributes $(K_1,\ldots K_m)$ conform a key of $\overline{R}$, which is called *expanded key*. □

In vertical partitions, the expanded key is projected in all sub-areas and each attribute of the original relation is projected in only one sub-area. We formalize the concepts of partition, area and sub-area as follows:

Definition 4.2 (horizontal partition). Given a relation R, its key expansion $\overline{R}$ and a set of conjunctive predicates $\{P_1,\ldots P_m\}$ over R, complete and disjoint (i.e. each tuple of R verifies one and only one predicate), a *horizontal partition* of R, denoted $\mathcal{HP}_{P_1 \ldots P_m}(R)$, is a set of selection views $\{R_1,\ldots R_m\}$, called *areas*, obtained applying the predicates $P_1,\ldots P_m$ to $\overline{R}$.

$$\mathcal{HP}_{P_1 \ldots P_m}(R) = \{R_1,\ldots R_m\} = \{\sigma_{P_1}(\overline{R}),\ldots \sigma_{P_m}(\overline{R})\}$$

We denote an area by a 4-uple <Name, Predicate, NumberTuples, KeyAccuracy> where *Name* is a name that identifies the area with respect to the relation, *Predicate* is the conjunctive predicate that defines the area (noted between square brackets), *NumberTuples* is the estimated number of tuples verifying the predicate and *KeyAccuracy* is the estimated accuracy of the key attributes of R. □

Definition 4.3 (vertical partition). Given an area R_i of relation R and n disjoint subsets of attributes of R $(S_1 \ldots S_n)$, a *vertical partition* of R_i, denoted $\mathcal{VP}_{S_1 \ldots S_n}(R_i)$, is a set of projection views $\{R_{i1},\ldots R_{in}\}$, called *sub-areas*, obtained projecting the subsets of attributes to R_i. Each attribute of R is projected in one and only one sub-area. The expanded key K of $\overline{R}$ is projected in all sub-areas.

$$\mathcal{VP}_{S_1 \ldots S_n}(R_i) = \{R_{i1},\ldots R_{in}\} = \{\pi_{K,S_1}(R_i),\ldots \pi_{K,S_n}(R_i)\}$$

We denote a sub-area by a 3-uple <Name, Attributes, Accuracy> where *Name* is a name that identifies the sub-area with respect to the relation, *Attributes* is the subset of attributes that defines the sub-area and *Accuracy* is the estimated accuracy of the sub-area. □

Areas and sub-areas are virtual relations (views) defined by the partitioning criteria (selection conditions and projection attributes), i.e. source relations are not physically fragmented and stored as a set of independent fragments. Source relations are kept unchanged in data sources and metadata describing areas and sub-areas (which was specified in Definition 4.2 and Definition 4.3) is stored at the DIS.

The following example illustrates the definition of partitions:

Example 4.10. Consider the *Students*(stid, name, interview, test, address, telephone) and *Marks*(stid, year, mark) relations presented in Example 4.1. The key expansion of relations are: $\overline{Students}$(Kstid, stid, name, interview, test, address, telephone) and $\overline{Notes}$(Kstid, Kyear, stid, year, mark), where {Kstid} and {Kstid,Kyear} are the expanded keys.

In this step the relations are partitioned in areas and sub-areas and metadata (number of tuples, accuracy values) is estimated. A possible partition of the *Students* relation can be:

- Area S_1; [stid < 101]; 6 tuples; key accuracy=0.50
  - Sub-area S_{11}; {stid, name, interview, test}; accuracy=0.50
  - Sub-area S_{12}; {address, telephone}; accuracy=0.25
- Area S_2; [stid ≥ 101]; 4 tuples; key accuracy=1.00
  - Sub-area S_{21}; {stid}; accuracy=1.00
  - Sub-area S_{22}; {name, interview, test, address, telephone}; accuracy=0.87

Analogously, a possible partition of the *Marks* relation can be:

- Area M_1; [year < 2004]; 1 tuple; key accuracy=0.00
  - Sub-area M_{11}; {stid, year, mark}; accuracy=0.00
- Area M_2; [year ≥ 2004 ∧ stid < 101]; 6 tuples; key accuracy=0.67
  - Sub-area M_{21}; {stid, year, mark}; accuracy=0.67
- Area M_3; [year ≥ 2004 ∧ stid ≥ 101]; 8 tuples; key accuracy=1.00
  - Sub-area M_{31}; {stid, year, mark}; accuracy=1.00 □

After partitioning a relation (horizontally and vertically), each cell of the relation belongs to a unique sub-area of a unique area, so we can represent partitions assigning different colors or patterns to each sub-area. Table 4.4 illustrates the partition of the *Students* relation introduced in previous example. As the expanded key belongs to all sub-areas, it is not colored and can be omitted in the graphical representation.

Kstid	stid	name	interview	test	address	telephone
21	21	María Roca	low	1.0	Carrasco	6001104
22	22	Juan Pérez	medium	.5	Coloniaa 1280/403	9023365
43	43	Emilio Gutiérrez	high	.8	Irigoitia 384	3364244
56	56	Gabriel García	low	.5	Propios 2145/101	
57	57	Laura Torres	medium	.7	Maldonado & Yaro	099628734
58	58	Raúl González	high	9	Rbla Rca Chile 1280/1102	4112533
101	101	Carlos Schnider	high	.9701	Copacabana 1210	094432528
102	102	Miriam Revoir	medium	.7945		9001029
103	103	A. Benedetti	low	.9146	Charrúa 1284/1	7091232
104	104	Luis López	high	.8220	Sixtina s/n	

Table 4.4 – Graphical representation of areas and sub-areas

In order to partition source relations, we can to use the Rakov's algorithm presented in Sub-section 3.1.3. But remember that the algorithm alternates horizontal and vertical splits and consequently, the obtained partition may not correspond a set of areas decomposed in sub-areas. For example, the partition of Figure 4.5a can be returned by the Rakov's algorithm. We have to restructure the partition according to Definition 4.2 and Definition 4.3.

Given any hybrid partition of a relation R consisting in a set of fragments $\{F_1,\ldots F_k\}$ verifying correctness rules, we can obtain another partition of R that follows Definition 4.2 and Definition 4.3 by further partitioning some of the fragments. In other words, we can obtain a set of sub-areas $\{S_{11},\ldots S_{1m_1},\ldots S_{n1},\ldots S_{nm_n}\}$ in which the sub-sets $\{S_{i1},\ldots S_{1i_1}\}$ vertically partition a certain area A_i, $1\leq i\leq n$, and the set of areas $\{A_1\ldots A_n\}$ horizontally partition R. We proceed as follows:

1. We obtain the set of selection predicates $P=\{P_1\ldots P_r\}$ that define the fragments $F_1\ldots F_k$, $r\leq k$. Note that r can be smaller than k since several fragments can have the same selection predicate.
2. While there exist in P two predicates P_i and P_j with a common sub-expression, i.e. $P_i \cap P_j \neq \emptyset$, we substitute P_i and P_j by $P_i \cap P_j$, $P_i - P_j$ and $P_j - P_i$. Since the predicates are conjunctions of simple inequalities on the attributes of R, this step will finish. We obtain a set of disjoint predicates $P'=\{P_1',\ldots P_n'\}$.
3. We define a set of areas $\{A_i,\ldots A_n\}$ one for each predicate in P'. Note that $\{A_i,\ldots A_n\}$ conforms a horizontal partition of R, i.e. it is disjoint because the predicates in P' are disjoint and it is complete because $\{F_1,\ldots F_k\}$ was complete and we do not loose sub-expressions of predicates.
4. We intersect each area A_i with the fragments $F_1\ldots F_k$, obtaining a set of sub-areas $\{S_{i1},\ldots S_{1i_1}\}$. Note that $\{S_{i1},\ldots S_{1i_1}\}$ conforms a vertical partition of A_i, i.e. it is complete and disjoint because $\{F_1,\ldots F_k\}$ was complete and disjoint.

> **Example 4.11.** Consider the hybrid partition of relation R shown in Figure 4.5a. Let P_i be the selection predicate of fragment F_i, $1\leq i\leq 7$. We define $X= P_3 \cap P_4$. Observe that $P_5=P_6=P_7$, $P_3= P_2 \cup X$ and $P_4= X \cup P_5$. Then, the set of disjoint predicates obtained in step 2 is $P'=\{P_1,P_2,X,P_5\}$, which define the areas $\{A_1, A_2, A_3, A_4\}$ shown in Figure 4.5b. Intersecting areas with the original fragments, we obtain the sub-areas $\{S_{11}, S_{21}, S_{22}, S_{31}, S_{32}, S_{41}, S_{42}, S_{43}, S_{44}\}$ shown in Figure 4.5b.

Then, any hybrid partition of a relation (satisfying correctness rules) can be restructured in areas and sub-areas. In particular, the partition produced by the Rakov's algorithm can be expressed in this way.

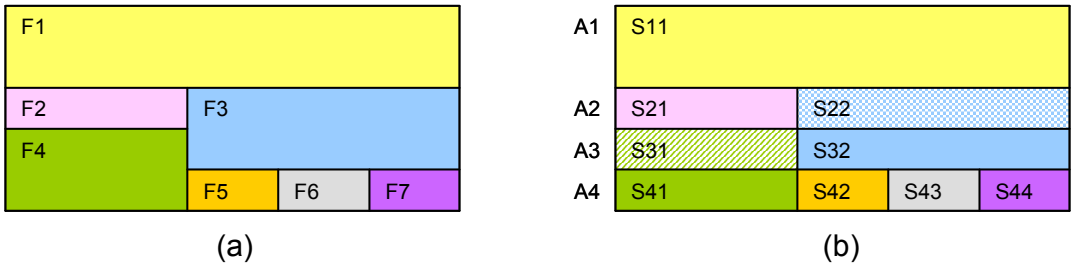

Figure 4.5 – Restructure of an arbitrary partition in areas and sub-areas: (a) partition before restructure and (b) partition after restructure

In order to partition source relations, we propose to use the Rakov's algorithm, however, areas and sub-areas can be also defined manually, using knowledge about source relations obtained from source administrators, third-party consultants, experts or users' feedback.

The Rakov's algorithm computes the accuracy of an area as the proportion of semantically correct cells. Such accuracy value (p) is the parameter for calculating the Gini indexes $G(v)=2p(1-p)$, which are used to compute accuracy homogeneity and consequently to choose the split that represents maximal gain in homogeneity (see Sub-section 3.1.3 for details). In other words, the partitioning algorithm can be parameterized in order to calculate other accuracy metric, providing a function for computing the accuracy of areas.

The partition obtained applying the Rakov's algorithm is restructured as previously explained. Metadata describing areas and sub-areas is also obtained from the partition, as follows:

- Areas are sequentially named as well as their sub-areas.
- The predicates defining areas and the sub-sets of attributes defining sub-areas are taken from the partition.
- The accuracy of a sub-area is estimated as the average of the accuracy of the cells of the sub-area. This estimation is also calculated by Rakov's algorithm.
- Accuracy of keys is estimated as the average of the accuracy of attributes that conform the key (accuracy of attributes is the accuracy of the sub-areas they belong because of accuracy homogeneity). Note that when all key attributes are projected in a unique sub-area, the accuracy of the key coincides with the accuracy of the sub-area.
- Finally, remember that the Rakov's algorithm partitions a sample of the source relation, so the number of tuples of an area is extrapolated from the number of tuples of the area in the sample, i.e. multiplying it by the ratio between relation size and sample size.

Partitioning is performed once (at DIS construction) or periodically, but it is separated from the evaluation of accuracy of user queries. Next sub-section describes how queries are rewritten in terms of partitions and its impact in accuracy evaluation.

4.2. Rewriting of user queries in terms of partitions

We propose to rewrite user queries in terms of areas of source relations. At the end of the sub-section we consider the alternative of rewriting user queries in terms of sub-areas and we explain why we discarded this option.

In order to express user queries in terms of the areas of source relations, we propose using classical query rewriting algorithms, as the Bucket algorithm [Levy+1996]. To this end, areas must be expressed as views of the global schema, following the local-as-view (LAV) approach. This sub-section explains the rewriting principle.

In the local-as-view (LAV) approach source relations are expressed as views of the global schema. As each area is a view over a source relation, it can be easily expressed as a view of the global schema by unfolding the view over the source relation, i.e. substituting the source relation with its definition in terms of the global schema.

The expression of areas in terms of the global schema is independent of user queries, i.e. it is done after partitioning source relations, but it is separated from the evaluation of accuracy of user queries.

Example 4.12. Continuing Example 4.10, the *Students* and *Marks* relations are expressed in terms of the global schema (relations *S* and *M*) as follows:

- Students(id, na, in, te, a, tp) ← S(id, na, nt, a, c, tp, in, te) ∧ c = 'Montevideo'
- Marks(id, y, m) ← M(id, y, m)

Remember the definition of their areas (expressed in Datalog-like notation):

- S_1(id, na, in, te, a, tp) ← Students(id, na, in, te, a, tp) ∧ id < 101
- S_2(id, na, in, te, a, tp) ← Students(id, na, in, te, a, tp) ∧ id ≥ 101
- M_1(id, y, m) ← Marks(id, y, m) ∧ y < 2004
- M_2(id, y, m) ← Marks(id, y, m) ∧ y ≥ 2004 ∧ id < 101
- M_3(id, y, m) ← Marks(id, y, m) ∧ y ≥ 2004 ∧ id ≥ 101

Substituting *Students* and *Marks* by their definitions we obtain the expression of areas in terms of the global schema:

- S_1(id, na, in, te, a, tp) ← S(id, na, nt, a, c, tp, in, te) ∧ c='Montevideo' ∧ id < 101
- S_2(id, na, in, te, a, tp) ← S(id, na, nt, a, c, tp, in, te) ∧ c='Montevideo' ∧ id ≥ 101
- M_1(id, y, m) ← M(id, y, m) ∧ y < 2004
- M_2(id, y, m) ← M(id, y, m) ∧ y ≥ 2004 ∧ id < 101
- M_3(id, y, m) ← M(id, y, m) ∧ y ≥ 2004 ∧ id ≥ 101 □

In order to rewrite user queries in terms of areas, the *Bucket* algorithm first create buckets, comparing query predicates with area predicates and then generates the candidate rewritings, checking query containment. The Bucket algorithm was described in Sub-section 3.2. The following example shows its use:

Example 4.13. Consider $UserQuery_2$ presented in Example 4.1, expressed in Datalog-like notation:

- $UserQuery_2$(id, y, n, m) ← S(id, n, nt, a, c, tp, in, te) ∧ M(id, y, m) ∧ y=2005

In order to rewrite $UserQuery_2$ in terms of S_1, S_2, M_1, M_2 and M_3 discussed in previous example, the following buckets are created: Buckets(S)={S_1,S_2} and Buckets(M)={M_2, M_3}. The area M_1 is not included in the second bucket because it is contradictory to the query predicate ('year < 2004' and 'year = 2005').

The following rewritings are generated, taking an area of each bucket:

- QR_1 (id, y, n, m) ← S_1(id, n, in, te, a, tp) ∧ M_2(id, y, m) ∧ y=2005
- QR_2 (id, y, n, m) ← S_2(id, n, in, te, a, tp) ∧ M_3(id, y, m) ∧ y=2005

Area S_1 is not combined with area M_3 because they are contradictory ('id < 101' and 'id ≥ 101'); analogously, area S_2 is not combined with area M_2. So, $UserQuery_2 \supseteq QR_1 \cup QR_2$.

We only considered the areas of the *Students* and *Marks* relations for reducing the size of the example, but in order to rewrite $UserQuery_2$ the areas of the *Classing* relation (areas C_1, C_2 and C_3 discussed in Example 4.2) must be also considered for being added to Bucket(S). Consequently, new query rewritings may result from combining C_1, C_2 and C_3 with M_2 and M_3. □

The number of rewriting grows polynomially with the number of areas. As proved in [Levy+1996], the intractability of rewriting algorithms is not in the number of relations but in the size of the query (which tends to be small) and only occurs when queries have multiple occurrences of the same relations.

We consider now the alternative of rewriting user queries in terms of sub-areas, explaining why we discarded it. Areas keep whole tuples of source relations. Therefore, when rewriting queries over areas, the rewriting algorithm joins whole tuples of sources relations (which is analogous to rewrite queries over source relations). However, sub-areas break tuples because they project some attributes of a relation. Firstly, we remark that there are extensions to the Bucket algorithm that consider the join of several relations of the same bucket in order to provide all required attributes, so query rewriting over sub-areas is possible. However, joining few attributes of a lot of relations seriously increase the risk of introducing semantic inconsistencies, i.e. building tuples that have no sense in real world. For example, we can return the name of a student and the telephone of another student, both having the same student id in different source databases. This risk is inherent to DIS but it extremely increases when we increase the number of joins (i.e. when we break tuples). In addition, rewriting algorithms avoid breaking tuples when possible for reducing this risk.

For this reason, we choose to rewrite user queries over areas and logically maintain sub-area metadata for calculating data accuracy. This is also the reason for partitioning source relations horizontally and then vertically instead of managing arbitrary hybrid partitions as in [Rakov 1998]. In next sub-section we evaluate data accuracy for each rewriting.

4.3. Estimation of data accuracy of query results

In order to estimate accuracy of areas and aggregate an accuracy value for the query result, we proceed in three sub-steps: (i) determining areas and sub-areas of the rewriting, (ii) estimating accuracy and key accuracy or the rewriting, and (iii) estimating the number of tuples of each area for performing the aggregation. Next sub-sections describe each sub-step.

4.3.1. Determination of the areas and sub-areas of a rewriting

Each rewriting is defined as the join of several areas, each area containing some (eventually one) sub-areas. The result of the join is one area, which satisfies the selection predicates of all input areas, and contains the union of all sub-areas having some of the projected attributes. Next example illustrates this.

Example 4.14. Figure 4.6 illustrates a query rewriting that joins three areas (A_1, A_2 and A_3), each one having several sub-areas (represented with different colors and patterns). The rewriting consists of an area (QR) with the union of the sub-areas that contain some of the projected attributes. Sub-area A_{12} does not belong to QR because none of its attributes are projected in the rewriting. □

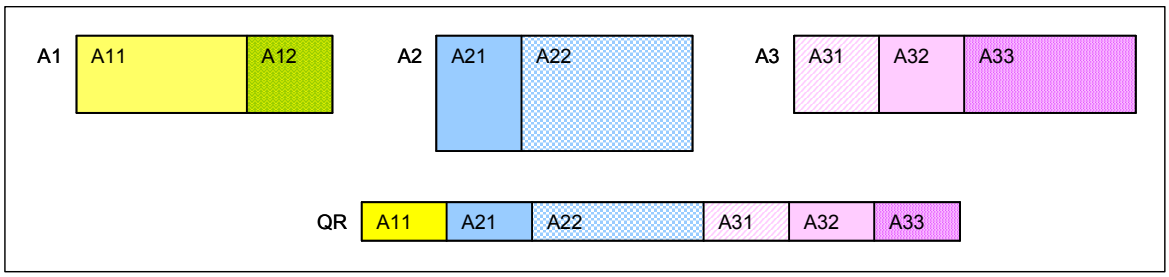

Figure 4.6 – Joining several areas

Each tuple returned by the rewriting will satisfy the predicates of all areas. So we build a unique area for the rewriting, as a conjunction of the predicates of the areas and the predicates of the rewriting. Note that predicates cannot be contradictory because the *Bucket* algorithm only returns contributive rewritings. Predicates that are less restrictive than other ones are not listed (e.g. 'year = 2005' is more restrictive than 'year ≥ 2004'.

The sub-areas of the rewriting are the union of the sub-areas of the input areas, intersecting attributes with the rewriting attributes. If the rewriting does not project any of the attributes of a sub-area, the sub-area is eliminated. The sub-areas of the rewriting conform a vertical partition of its unique area. The satisfaction of the completeness property is straightforward because all attributes of the rewriting belong to some sub-area of the inputs areas. Regarding the disjointness property, a problem can arise with natural join because, join attributes belong to two input sub-areas (which is necessary for performing the join) but are projected only once in the rewriting. The following example illustrates this case:

Example 4.15. Consider the rewriting QR_1 of Example 4.13:

$$QR_1\ (id, y, n, m) \leftarrow S_1(id, n, in, te, a, tp) \wedge M_2(id, y, m) \wedge y=2005$$

The *id* variable is used in areas S_1 and M_2, conforming a natural join. The *id* variable represents an attribute of S_1 (which is projected in a sub-area of S_1) and an attribute of M_2 (which is projected in a sub-area of M_2). But it also represents an attribute of the rewriting, which must be projected in a unique sub-area. □

Each natural join attribute must be projected in a unique sub-area of the rewriting in order to satisfy the *disjointness* property required for partitions. To this end, we create a new sub-area for the attribute, whose accuracy value will be calculated as an average of the accuracy of both sub-areas, as will be explained in next sub-section. After calculating accuracy, sub-areas having the same accuracy value will be fusioned.

Example 4.16. Continuing Example 4.13, areas and sub-areas for QR_1, and QR_2 are calculated as follows: One area is created for each rewriting, QR_1 and QR_2 respectively, labeled with the input areas and the rewriting predicates. Rewriting QR_1 joins areas S_1 and M_2, whose sub-areas are S_{11}, S_{12} and M_{21}. Sub-areas QR_{11} and QR_{12} result from intersecting attributes of sub-areas S_{11} and M_{21} with rewriting attributes; the join attribute is separated in sub-area QR_{13}. No attribute of sub-area S_{12} is projected. Sub-areas of QR_2 are calculated analogously. We obtain:

- Area QR_1 {stid < 101 ∧ year = 2005}; ? tuples; key accuracy = ?; inputs: S_1 and M_2
  - Sub-area QR_{11} {name}; accuracy = ?; input: S_{11}
  - Sub-area QR_{12} {year, mark}; accuracy = ?; input: M_{21}
  - Sub-area QR_{13} {stid}; accuracy = ?; inputs: S_{11} and M_{21}
- Area QR_2 {stid ≥ 101 ∧ year = 2005}; ? tuples; key accuracy = ?; inputs: S_2 and M_3
  - Sub-area QR_{21} {name}; accuracy = ?; input: S_{22}
  - Sub-area QR_{22} {year, mark}; accuracy = ?; input: M_{31}
  - Sub-area QR_{23} {stid}; accuracy = ?; inputs: S_{21} and M_{31}

Input areas or sub-areas where also registered for understanding purposes. □

Next sub-section explains how to calculate accuracy values and Sub-section 4.3.3 discusses the estimation of the number of tuples.

4.3.2. Accuracy calculation

If partitions are quite well defined, they should reasonably preserve the accuracy of sub-areas, being insensible to projection attributes and selection predicates, because of accuracy homogeneity. Regarding joins, as a tuple is build from two input tuples, the accuracy of resulting sub-areas may depend on both input areas. Specifically, accuracy may be propagated differently depending on the accuracy factor used.

Remember that when evaluating semantic correctness metrics, if the key of a tuple is not accurate (does not correspond to a real-world entity) the whole tuple (all cells) are inaccurate. In other words, semantic correctness metrics evaluate if an attribute corresponds to the real world object represented by the key. For example, if a student identification number does not exists (e.g. stid 21), the data associated to such student id is incorrect too (i.e. 'María Roca' cannot be her name, 'Carrasco' cannot be her address, etc.). However, when evaluating syntactic correctness or precision, accuracy of cells does not depend on keys, for example, syntax constraints, belonging to a range or decimal precision can be checked independently of the key values.

Then, when joining two relations (two areas), the accuracy of the cells in the result is calculated differently for semantic accuracy. The cells in the result that are semantically correct are those that were obtained from a correct value with two correct keys. When one of the keys is incorrect, the complete tuple is incorrect too. So, the accuracy of a sub-area in the join result is calculated as the accuracy of the input sub-area multiplied by the accuracy of the keys of the other areas. However, when evaluating syntactic correctness or precision, the accuracy of cells in the result is the accuracy of the cell in the input relation.

In summary, we proceed as follows:

- For syntactic correctness and precision factors: Accuracy of a sub-area is estimated as the accuracy of the input sub-area.
- For semantic correctness: Accuracy of a sub-area is estimated as the accuracy of the input sub-area multiplied by the key accuracy of the other input areas.

For sub-areas containing join attributes (that were separated in previous step), accuracy is calculated separately from each input sub-area and the average is taken. If some sub-areas have the same accuracy value, they are fusioned in a unique sub-area. Finally, key accuracy is calculated multiplying accuracy of keys of all input areas.

Example 4.17. Continuing Example 4.16, we estimate semantic correctness of sub-areas, obtaining:

- Area QR_1 {stid < 101 ∧ year = 2005}; ? tuples; key accuracy = 0.33 (0.50 * 0.67); inputs: S_1 and M_2
  - Sub-area QR_{11} {name}; accuracy = 0.33 (0.50 * 0.67); input: S_{11}
  - Sub-area QR_{12} {year, mark}; accuracy = 0.33 (0.67 * 0.50); input: M_{21}
  - Sub-area QR_{13} {stid}; accuracy = 0.33 (average {0.50 * 0.67, 0.67 * 0.50}); input: S_{11} and M_{21}
- Area QR_2 {stid ≥ 101 ∧ year = 2005}; ? tuples; key accuracy = 1.00 (1.00 * 1.00); inputs: S_2 and M_3
  - Sub-area QR_{21} {name}; accuracy = 0.87 (0.87 * 1.00); input: S_{22}
  - Sub-area QR_{22} {year, mark}; accuracy = 1.00 (1.00 * 1.00); input: M_{31}
  - Sub-area QR_{23} {stid}; accuracy = 1.00 (average {1.00 * 1.00, 1.00 * 1.00}); inputs: S_{21} and M_{31}

Sub-areas {QR_{11}, QR_{12}, QR_{13}} and {QR_{22}, QR_{23}} have the same accuracy value, so they are fusioned in sub-areas QR_{11} and QR_{22} respectively, obtaining:

- Area QR_1 {stid < 101 ∧ year = 2005}; ? tuples; key accuracy = 0.33; inputs: S_1 and M_2
  - Sub-area QR_{11} {stid, year, name, mark}; accuracy = 0.33
- Area QR_2 {stid ≥ 101 ∧ year = 2005}; ? tuples; key accuracy = 1.00 (1.00 * 1.00); inputs: S_2 and M_3
  - Sub-area QR_{21} {name}; accuracy = 0.87
  - Sub-area QR_{22} {stid, year, mark}; accuracy = 1.00 □

The accuracy of each rewriting is aggregated as the average of accuracy of cells, i.e. weighting the average of accuracy of sub-areas by the number of attributes projected in the sub-area.

Example 4.18. Continuing Example 4.17, the accuracy of QR_1 is 0.33 (the accuracy of its unique sub-area) and the accuracy of QR_2 is 0.97 (0.87 * 1 + 1.00 * 3). □

Next sub-section explains the estimation of the number of tuples of each area.

4.3.3. Selectivity estimation

As the answer to a query can be built as the union of several rewritings, an accuracy value for the query is aggregated as the weighted sum of the accuracy of rewritings (weights are the number of tuples). To this end, we need to estimate how many tuples has each rewriting.

We propose to estimate the selectivity of rewritings in order to estimate their numbers of tuples. Any of the techniques discussed in Sub-section 3.3 can be used for estimating selectivity. In most cases, joins correspond to the equality between a primary key and a foreign key, so simple histograms are adequate for the estimation. However, other commercial algorithms used for query optimization or even statistics of previous execution of the same or similar queries can be used as estimation. In particular application scenarios, selectivity can be estimated by experts. Our approach is independent of the estimation strategy used but of course the obtained accuracy value depends on it.

We define selectivity as follows:

Definition 4.4 (selectivity). Given a query rewriting QR over a set of areas {R_1,…R_k}, the *selectivity* of QR, denoted *sel*(QR), is the proportion of tuples of the Cartesian product of the areas R_1,…R_k that verify the rewriting predicates. If the query rewriting has no selection (nor join) predicates, then all tuples are returned and thus selectivity is 1. □

The number of tuples of the rewriting is estimated multiplying the number of tuples of input areas by the selectivity of the rewriting.

Example 4.19. Continuing Example 4.18, consider that selectivity of QR_1 and QR_2 is estimated as 1/9 and 1/8 respectively. The complete metadata of the partition is:

- Area QR_1 {stid < 101 ∧ year = 2005}; 4 (6 * 6 * 1/9) tuples; key accuracy = 0.33; inputs: S_1 and M_2
  - Sub-area QR_{11} {stid, year, name, mark}; accuracy = 0.33
- Area QR_2 {stid ≥ 101 ∧ year = 2005}; 4 (4 * 8 * 1/8) tuples; key accuracy = 1.00 (1.00 * 1.00); inputs: S_2 and M_3
  - Sub-area QR_{21} {name}; accuracy = 0.87
  - Sub-area QR_{22} {stid, year, mark}; accuracy = 1.00

The global accuracy value for $UserQuery_2$ is calculated weighting the accuracy of each rewriting by its number of tuples, obtaining (4 * 0.33 + 4 * 0.97) / (4 + 4) = 0.65. Note that the same value can be obtained weighting the accuracy of sub-areas of all areas by their number of cells, i.e. (0.33 * 16 + 0.87 * 4 + 1.00 * 12) / (16+4+12) = 0.65. □

Next sub-section discusses the implementation of this evaluation strategy in an accuracy evaluation algorithm.

4.4. Reuse of the quality evaluation framework

In this sub-section we describe how the quality evaluation framework introduced in Chapter 3 is reused for data accuracy evaluation. We firstly describe the construction of quality graphs and then we present an accuracy evaluation algorithm.

4.4.1. Construction and adornment of quality graphs

Once the query rewritings have been generated, a quality graph can be build for representing the calculation of the user query as the union of several (possibly one) rewritings.

The graph is build as follows:

- The user query is represented by a target node
- Source relations are represented by source nodes.
- Areas of source relations are represented by activity nodes, called *area nodes*. Edges link each area node with the corresponding source node.
- Rewritings are represented by activity nodes, called *rewriting* nodes. Edges link each rewriting node with the ones representing the areas referenced by the rewriting.
- The union of rewritings (eventually only one rewriting) is represented by an activity node, called the *union node*. This activity is predecessor of the target node and successor of all rewriting nodes.

Figure 4.7 shows the quality graph for $UserQuery_2$ expressed as the union of rewritings QR_1 and QR_2, computed in Example 4.13.

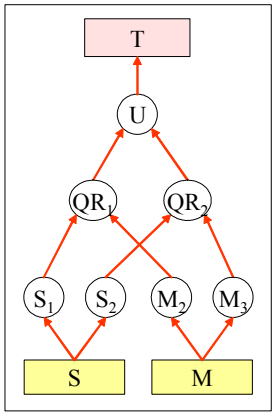

Figure 4.7 – Quality graph representing the union of two query rewritings

Several properties, necessaries in the evaluation of data accuracy, adorn the quality graph and are input to the accuracy evaluation algorithm. Metadata describing areas and sub-areas is contained in the following property:

- *Areas*: It is an ordered list of records (representing areas) composed of the following fields:
  - *Predicate*: It is a conjunctive predicate defining the area
  - *Cardinality*: It is the estimated number of tuples in an area (verifying the area predicate)
  - *Key accuracy*: Accuracy of key attributes
  - *Sub-areas*: It is an ordered list of sub-areas, represented as pairs < attributes, accuracy> composed of: the set of attributes that defines the sub-area and its accuracy value.

This property is associated to source nodes (containing one record for each area) and area nodes (containing a unique record for the corresponding area). Rewriting nodes have associated further properties:

- *Predicate*: It is a conjunctive predicate defining the rewriting predicate
- *Projected attributes*: It is the set of attributes projected by the rewriting
- *Join attributes*: It is the set of natural join attributes of the rewriting
- *Consider key*: It is a Boolean flag indicating if key accuracy must be considered in accuracy propagation (This flag will be true for the propagation of semantic correctness and false for the propagation of syntactic correctness and precision).
- *Selectivity*: It is the selectivity of the rewriting

The evaluation algorithm calculates the following property:

- *Actual accuracy*: It is an estimation of the actual accuracy of data outgoing a node (*Aaccuracy* for short).

The algorithm also computes areas and sub-areas of the rewritings (and therefore associates the *areas* property to rewriting and union nodes). Next sub-section describes the evaluation algorithm.

4.4.2. Accuracy evaluation algorithm

In this sub-section we propose a basic algorithm for evaluating data accuracy. It follows the principle described in Sub-section 4.3, proceeding in three steps: (i) for each rewriting node, it creates an area, calculates its sub-areas and calculate property values (predicates, cardinality, key accuracy and accuracy of sub-areas); (ii) for the union node, it inserts areas of all rewriting nodes; and (iii) it aggregates accuracy values for all data edges. The pseudocode is sketched in Algorithm 4.2.

The first step performs two loops through area nodes incoming a rewriting. In the first loop, the *key accuracy* property is calculated multiplying the accuracy of input areas, the *cardinality* property is calculated multiplying rewriting selectivity by the cardinality of input areas and the *predicate* property is calculated as the conjunction of predicates of input areas and the predicate of the rewriting. The *insert* function of the *Predicate* class adds a new predicate as a new conjunction and also eliminates useless expressions (e.g. those that are less restrictive than other ones). In the second loop, the *sub-areas* property is calculated, inserting sub-areas of all input areas. This must be done in a separate loop because key accuracy (calculated in the first loop) is used for calculating accuracy of sub-areas in the case the *ConsiderKey* flag is switched on. After adding sub-areas to the unique area of the rewriting node, the *intersectAttributes* function intersects attributes of sub-areas with those projected by the rewriting, eventually eliminating empty sub-areas. Then, natural join attributes are separated in new sub-areas, calculating their accuracy as an average of accuracy of all sub-areas where they are contained. This is done by the *separateJoinAttributes* function. Finally, the *fusionSubAreas* function fusions sub-areas having the same accuracy value.

The areas created for each rewriting are added to a list (*Uareas*), so the second step only needs to set the list as value of the *areas* property of the union node.

In the last step, for each source or activity node, the *aggregateAccuracy* function performs a weighted sum of the accuracy of sub-areas, where weights are obtained multiplying the number of attributes of the sub-area by the cardinality of the area. The obtained value is associated to all data edges outgoing the node.

In next sub-section we deal with data accuracy improvement. We present a direct application of the approach for discarding the areas (or sub-areas) that cause overdrawing accuracy expectations.

```
FUNCTION ActualAccuracyPropagation (G: QualityGraph) RETURNS QualityGraph
   ListOfAreas Uareas; // will contain all areas, which will be associated to the union node

   FOR EACH rewriting node R in G DO // Set areas to writing nodes
      Area area;
      // First loop; sets predicate, cardinality and key accuracy
      INTEGER K= 1; // will contain the product of accuracies of all keys
      INTEGER card = G.getPropertyValue(R,"Selectivity");
      Predicate pred = G.getPropertyValue(R,"Predicate");
      FOR EACH predecessor A of R in G DO
         Area auxarea = G.getPropertyValue(A,"Areas").getFirstElement();
         K= K * area.getKeyAccuracy();
         card = card * area.getCardinality();
         pred.insert (area.getPredicate());
      ENDFOR;
      area.setKeyAccuracy(K);
      area.setCardinality(card);
      area.setPredicate(pred);
      // Second loop; sets sub-areas
      FOR EACH predecessor A of R in G DO
         Area auxarea = G.getPropertyValue(A,"Areas").getFirstElement();
         IF (auxarea.getConsiderKey() == TRUE)
            auxarea.updateSubAreasAccuracy(K);
         area.addSubAreas(auxarea.getSubAreas());
      ENDFOR;
      area.IntersectAttributes(G.getPropertyValue(R,"ProjectedAttributes"));
      area.SeparateJoinAttributes(G.getPropertyValue(R,"JoinAttributes"));
      area.FusionSubAreas();
      G.addPropertyValue(R,"Areas",{area});
      Uareas.add (area);
   ENDFOR;

   Node U = union node of G; // Set areas to the union node
   G.addPropertyValue(U,"Areas",Uareas);

   FOR EACH source and activity node A of G DO // Set accuracy aggregations to outgoing edges
      ListOfAreas areas= G.getPropertyValue(A,"Areas");
      INTEGER value= aggregateAccuracy (areas);
      FOR EACH data edge e outgoing A in G
         G.addProperty(e,"ActualAccuracy",value);
      ENDFOR;
   ENDFOR;
   RETURN G;
END
```

Algorithm 4.2 - Basic algorithm for propagating accuracy actual values

5. Accuracy improvement

In this section we discuss some base improvement actions for enforcing data accuracy when user accuracy expectations cannot be satisfied. Accuracy expectations correspond to upper bounds for the accuracy of result data. Remember that our evaluation approach groups result data in sub-areas having homogeneous accuracy. This means that we cannot assure that all cells in the result satisfy user expectations. Conversely, we can deliver data that, in average, satisfies user expectations.

The proposal for accuracy improvement is simple: *filtering "portions" of the query result having low accuracy*. In this sub-section we discuss several ways and several moments for performing such filtering, which depend on user expectations. Specifically, three types of accuracy expectations can be expressed:

- Accuracy of groups of cells, in average, should be lower than a threshold
- Accuracy of groups of tuples, in average, should be lower than a threshold
- Accuracy of the whole result should be lower than a threshold

The first type of condition is the most restrictive. It corresponds to users that cannot tolerate tuples having inaccurate values, even if other cells have high accuracy values. Furthermore, accuracy expectations may concern only certain attributes, for example, users may require telephone numbers to be accurate, ignoring the accuracy of other attributes. As we partition query result in sub-areas having homogeneous accuracy, this type of condition suggests the filtering of sub-areas having low accuracy. If the condition involves only certain attributes, only the sub-areas containing those attributes are candidate to be filtered. Note that such filtering can be done in an early phase of the evaluation, i.e. when adding areas to buckets. We use the term *selective rewriting* for naming this improvement action.

The second type of condition tolerates some attributes with low accuracy if the accuracy aggregation for the tuple is high enough. Certain users may accept receiving tuples that contain errors in some attributes if other cells are accurate. For example, when several attributes contain alternative ways of contacting customers (address, telephone, email…) errors in some attributes may be tolerated if the others have high accuracy values. This type of condition suggests the filtering of areas instead of sub-areas, i.e. aggregating an accuracy value from the accuracy of sub-areas and comparing it with user expectations. Conversely to the filtering of sub-areas, the filtering of areas must be done after computing rewritings, because the aggregation may concern sub-areas of several source relations.

The last type of condition does not care about the accuracy of attributes or tuples but expresses that the whole result should achieve a certain level of accuracy. Note that query results consisting of tuples with very low accuracy and other tuples with high accuracy, may be accepted. This corresponds to users that want as much result data as possible, without neglecting accuracy. For example, a user may send advertising letters to their customers, tolerating up to 5% of mail return due to errors in contact information; this means that he wants to send the maximum of letters while he can assure a certain benefice. In order to satisfy this kind of constraints, we should deliver as many data as possible without descending to much their accuracy. This suggests ordering areas according to their accuracy and aggregating accuracy values incrementally, stopping when accuracy expectations are no longer satisfied. Furthermore, we can incrementally deliver data to users, allowing them to stop the delivery when they have enough data or data has too many errors. This strategy can be performed only after aggregating accuracy of all candidate rewritings.

Three improvement actions were motivated: (i) selective rewriting of user queries, (ii) filtering of rewritings, and (iii) incremental delivery of results. In the following, we describe each action.

Selective rewriting consists in generating query rewritings that satisfy a certain quality requirement, specifically: 'accuracy of all sub-areas must be higher than a threshold'. We propose modifying the rewriting algorithm, for excluding from buckets, the areas that have a "concerned" sub-area with lower accuracy. By *concerned* with means that the sub-area provides attributes for solving the query, i.e. the query will project some of its attributes. As the accuracy of sub-areas was pre-calculated during partitioning, the implementation of this strategy is straight-forward.

The *filtering of rewritings* is also straight-forward. It can be done at the third step of our approach, i.e. when aggregating accuracy of rewritings (see Sub-section 4.3.2).

For the *incremental delivery of results*, we propose aggregating accuracy of rewritings and ordering them according their accuracy. A simple loop and two variables implement this improvement action, as sketched in Algorithm 4.3. The algorithm receives a list of areas (of rewritings) and an accuracy expected value and returns a

list of areas ordered by accuracy (descendent order). The returned areas are those that can be conveyed to users satisfying accuracy expectations; conveying more areas means not longer achieving user expectations.

Note that the estimation of rewriting selectivity is only useful for the third improvement action; the other actions filter individual sub-areas and rewritings.

```
FUNCTION IncrementalDelivery (inAreas: ListOfAreas, Eaccuracy: INTEGER) RETURNS ListOfAreas
   areas.sortByAccuracy();
   ListOfAreas outAreas;

   Area A = areas.getFirst();
   INTEGER acc= A.getAccuracy() * A.getCardinality(); // partial sum of accuracies * cardinalities
   INTEGER card = A.getCardinality(); // partial sum of cardinalities

   WHILE acc/card > Eaccuracy;
      outAreas.add(A);
      A = areas.getNext();
      acc = acc + A.getAccuracy() * A.getCardinality();
      card = card + A.getCardinality();
   ENDWHILE;
   RETURN areas;
END
```

Algorithm 4.3 – Incremental delivery of data

The following example summarizes the proposed improvement actions.

Example 4.20. Consider the four rewritings illustrated in Figure 4.8 (QR_1, QR_2, QR_3 and QR_4). Accuracy values are written under sub-areas and aggregated values for areas are written, in bold, under area name. Consider the following conditions:

- C_1) Accuracy of sub-areas should be greater than 0.75
- C_2) Accuracy of areas should be greater than 0.75
- C_3) Accuracy of query result should be greater than 0.75

Condition C_1 is only satisfied by QR_2. Furthermore, QR_2 is the unique rewriting that should be generated. For example, as A_{12} does not satisfy the condition, the area containing A_{12} is deleted from the corresponding bucket and therefore, the rewriting QR_1 is not generated.

Condition C_2 is satisfied by QR_1 and QR_2. Although sub-area A_{12} has a lower accuracy value, the accuracy aggregation for the rewriting is acceptable. For that reason, filtering must occur after accuracy aggregation.

In order to verify condition C_3, rewritings are ordered according to their accuracy, obtaining the following order: $QR_2 > QR_1 > QR_3 > QR_4$. We incrementally aggregate accuracy of such rewritings. For $\{QR_2\}$ accuracy is 0.90. For $\{QR_2, QR_1\}$ accuracy is (0.90 * 20 + 0.80 * 10) / (20 + 10) = 0.86. Analogously, for $\{QR_2, QR_1, QR_3\}$ accuracy is 0.80. For $\{QR_2, QR_1, QR_3, QR_4\}$ accuracy is 0.70, which does not satisfy the condition; so QR_4 is eliminated. □

The most restrictive are the conditions, the smallest is the result. Accuracy expectations should be balanced with completeness expectations for avoid filtering too much data and conveying a representative set of tuples to users. The trade-off among data accuracy and other quality factors is discussed, in Chapter 6, as future work.

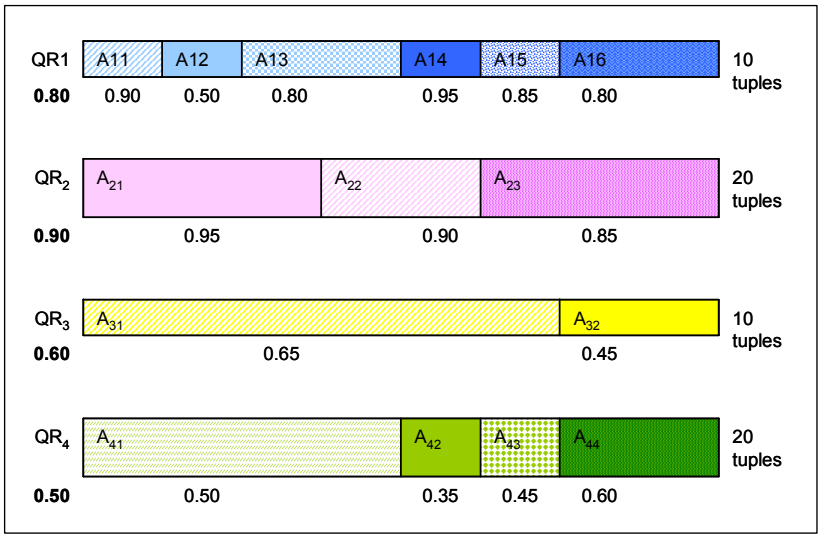

Figure 4.8 – Filtering areas and sub-areas

In terms of the elementary improvement actions proposed in Sub-section 4.4 of Chapter 3, the improvement actions proposed in this section correspond to deleting portions of the quality graph, i.e. invocating the *replaceSubGraph* action. In addition, the proposed improvement actions can be combined with other actions not treated in this thesis, for example, error correction techniques, in order to implement more complex improvement strategies for specific DISs.

6. Conclusion

In this chapter we dealt we data accuracy evaluation and enforcement topics.

Based on the partitioning algorithm proposed in [Rakov 1998], we proposed the partitioning of source relations according to data accuracy and the propagation of such partitions to query results in order to better describe the distribution of inaccuracies. User queries are rewritten in terms of partitions and accuracy values are aggregated for the different rewritings. The query result consists in the union of tuples returned by rewritings. The number of tuples is calculated, estimating the selectivity of the rewriting, in order to aggregate an accuracy value for the query result.

We presented a basic algorithm for data accuracy evaluation, which implements the proposal. Compared to existing proposals for *a priori* evaluation, our algorithm explicitly indicates the areas that have lower accuracy, which allows filtering data not satisfying accuracy expectations. The precision of the estimation relies on the techniques used for measuring accuracy of source relations, partitioning them and estimating the selectivity of rewritings. We do not tested different techniques in order to suggest the utilization of specific ones. This topic is suggested as perspective in Chapter 6.

We also proposed some basic improvement actions, for filtering data with low accuracy in order to satisfy several types of accuracy expectations. Other improvement strategies can be analyzed and combined to these ones in order support more complete improvement strategies. The partitioning mechanism and the evaluation algorithm proposed in this section may help in the analysis. The proposal can be used at different phases of the DIS lifecycle (e.g. at design, production or maintenance phases), either for communicating data accuracy to users, comparing data sources, checking the satisfaction of user accuracy expectations or analyzing improvement actions for enforcing data accuracy. Chapter 5 illustrates the use of the approach in a real application.

Chapter 5. Experimentation and Applications

This chapter presents our experimentation results.
We describe several applications and the use of the quality evaluation framework for evaluating data quality in such applications.
We describe a prototype of a Data Quality Evaluation tool, DQE, which implements the framework and allows the execution of quality evaluation algorithms.
The prototype was used for evaluating data freshness and data accuracy in several application scenarios.

1. Introduction

Chapters 3 and 4 described our proposal for the evaluation of data freshness and data accuracy in data integration systems. In this chapter we illustrate its practical use in real applications. To this end, we have developed a prototype of an evaluation tool, called DQE, which implements the proposed quality evaluation framework. The tool allows displaying and editing the framework components as well as executing quality evaluation algorithms.

The prototype was used for evaluating data freshness and data accuracy in several application scenarios in order to validate the approach. Specifically, we describe three applications: (i) an adaptive system for aiding in the generation of mediation queries, (ii) a web warehousing application retrieving movie information, and (iii) a data warehousing system managing information about students of a university. We briefly describe each application, we model it in DQE (quality graphs, properties, etc.) and we explain the evaluation of data freshness and data accuracy for them.

The goal of our experimentation is twofold. Firstly, we want to validate the approach in real applications, analyzing the practical difficulties of modeling different DISs as quality graphs, setting property values and instantiating evaluation algorithms for them. Secondly, we want to test the execution of evaluation algorithms for such quality graphs. For the first application (mediation application) we also tested the connection of the tool with other DIS design modules, communicating via a metadata repository.

In addition, we describe some tests for evaluating performance and limitations of the tool. To this end, we generated some data sets (quality graphs adorned with property values) and we executed a quality evaluation algorithm over each graph. The test results allow affirming that the tool can be used for large applications (modeling hundreds of graphs with hundreds of nodes each).

The following sections describe our experimentations: Section 2 presents the DQE tool, describing its functionalities, architecture and interface. Section 3 uses DQE in three application scenarios, describing the evaluation experiments performed in each application and their results. Section 4 presents the performance tests, describing the generation of test data sets, the tests themselves and the obtained results. Finally, section 5 concludes.

2. Prototype

Most of the functionalities of the framework for data quality evaluation described in previous chapters have been implemented in a quality auditing tool called *DQE* (Data Quality Evaluation). The prototype was implemented in Java (JDK 1.4) and manages persistence via an Oracle® database (Oracle 10g Enterprise Edition Release 10.1.0.2.0). The tool allows displaying and editing the framework components as well as executing quality evaluation algorithms. Next sub-sections describe the tool main characteristics; design and implementation features are treated in Annex 1.

2.1. Functionalities

The main functionalities of the tool are:

- *Storage of framework components*: The framework components (data sources, data targets, quality graphs, properties and algorithms), auxiliary components that ease the manipulation of framework (e.g. sessions, groups of nodes and edges) and graphical components, are all stored in a relational database, called *metabase*. The metabase contains global catalogues for the various components and maintains association lists among components. The goal of the metabase is threefold: it provides persistency, it allows reusing components previously defined, and it serves as communication media for sharing data with other DIS design tools (in particular with the tool managing user profiles, which stores user expected quality values) and statistical tools (which estimate and store property values).
- *Management of sessions*: A session represents a concrete application scenario and encloses the framework components for that scenario, i.e. data sources, data targets, quality graphs, properties and algorithms. Several sessions can be stored in the metabase but the tool only manages one session at a time. Among the functionalities of management of sessions, the tool includes: the creation, loading, storing and deletion of sessions, the creation, loading, storing and deletion of components. Most of the components (data sources, data targets, properties and algorithms) are stored in global catalogues of the metabase and can be loaded in several sessions. Quality graphs, however, are associated to a unique session.
- *Management of properties*: An editor of properties allows the creation, modification and deletion of properties from the global catalogue and their inclusion/removal from the current session. Information describing properties include: a code (auto-generated), a name (globally unique, used by the user for identifying the property), a description (natural language description of the property semantics and units), a type (feature or measure) and the domain data type (e.g. *integer*). In the case of measure properties, the dimension and factor and also required (e.g. the *Currency* metric corresponds to the *Currency* factor of the *Freshness* dimension).
- *Management of sources and targets*: Sources and targets are managed in the same way. A simple editor allows the creation, modification and deletion of sources/targets from the global catalogue and their inclusion/removal from the current session. Information describing sources/targets include: a code (auto-generated) and a description. Sources may have associated some property values (e.g. actual quality values, availability, access cost); targets may also have associated some property values (e.g. expected quality values, workload). The editor also allows associating and dissociating properties to sources and targets as well as changing property values.
- *Management of algorithms*: A simple interface allows the inclusion and deletion of algorithms from the global catalogue and their inclusion/removal from the current session. An algorithm is identified by a name (globally unique) and has a description and a reference to a Java class that implements it. The current version does not provide a graphical interface for implementing an algorithm; algorithm code should be produced using a text editor or a Java development environment. However, the tool provides a method for dynamically adding new algorithms (automatically binding arguments and properties) without closing the session.
- *Visualization and edition of quality graphs*: Quality graphs are the most important components managed by the application and consequently, a sophisticated display utility, called *graph viewer*, allows visualizing the graphs and their components (nodes, edges, labels). The graph viewer allows visualizing nodes and edges, moving them in order to better analyze the graph topology, showing or hiding property values, grouping nodes and edges, coloring nodes and edges according to different criteria and zooming. The graph viewer also allows adding and removing nodes and edges to a quality graph, and changing property values.
- *Association of properties to nodes and edges of a quality graph*: Nodes and edges are grouped in order to associate properties to them (for example: we can define the group *MaterializedActivities* and associate the property *RefreshFrequency* to all nodes in such group). The tool provides functionalities for creating and deleting groups of nodes and edges, inserting and deleting nodes and edges to a group and associating and dissociating properties from a group. Source and target nodes also inherit the properties of the corresponding source or target. As a node or edge can belong to several groups, they can have associated a same property several times, but a unique value is set.

- *Execution of an algorithm*: There are two ways of executing an algorithm. The first one consists in selecting a quality graph and invoking an algorithm (selected from the session algorithms) for the graph. The second one consists in invoking an algorithm for all the quality graphs of the session. The execution of an algorithm in a quality graph is performed in a separate thread, so other functionalities can be invoked during the execution of an algorithm. The execution of an algorithm on different graphs is parallelized but the execution of several algorithms on a same graph is sequential.
- *Visualization of results*: The graph viewer is automatically refreshed after the execution of an algorithm, showing the new property values (corresponding to the evaluated quality factor). The auditing tool allows coloring nodes that do not achieve quality expectations, identifying and coloring critical paths, changing property values in order to test alternative configurations and re-executing the evaluation algorithms to see the effects of the changes.

2.2. Architecture

The architecture is structured in layers and the communication among layers is done via interfaces in order to standardize the access to each layer. The design of the graphical interface follows the *model-view-controller* pattern, implemented in three different layers (*View*, *Logic* and *Model*). The persistency in a relational database (Oracle 10g) is implemented in the *DataAccess* layer. Exception management and other utility functions are accessed via the *Utilities* transversal layer. Figure 5.1 shows the layers of the architecture and the interaction among them, as well as the most relevant packages that compose each layer.

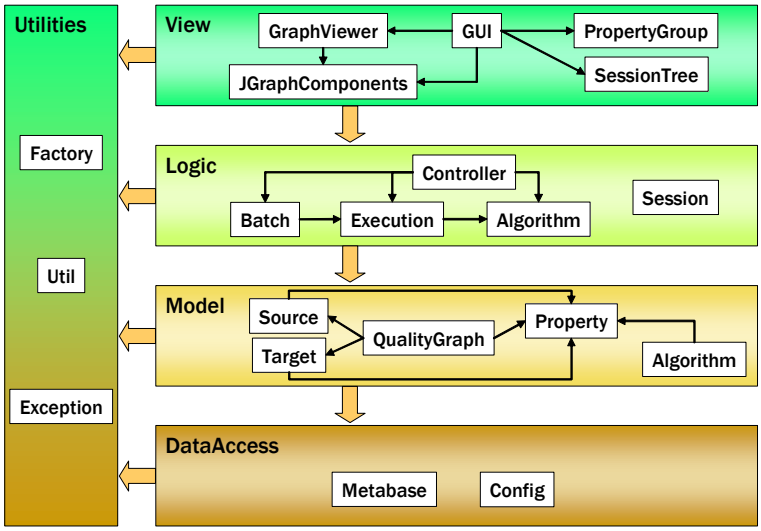

Figure 5.1 – Architecture of the prototype

The *View layer* manages the graphical representation. The main packages are: *GUI* (controls the graphical user interface), *GraphViewer* (displays and allows the creation and edition of quality graphs), *JGraphComponents* (manages graph components (nodes, edges, labels) and their graphical properties), *SessionTree* (manages session components) and *PropertyGroup* (manages the properties and groups associated to nodes and edges).

Logic layer: Implements the application logic, bringing methods for managing sessions and executing algorithms. The main packages are: *Controller* (acts as a bridge between graphical interface and data model), *Session* (manages sessions), *Algorithm* (manages the collection of algorithms, their arguments, preconditions and implemention), *Execution* (controls the execution of algorithms) and *Batch* (controls batch execution of algorithms over all quality graphs).

Model layer: Manages the data model. Tha main packages contain the representation of the framework components: *Source*, *Target*, *QualityGraph*, *Property* and *Algorithm*.

DataAccess layer: Persists the data model in a relational database (*Metabase* package) and interacts with configuration files (*Config* package).

Utilities layer: Brings basic services to the other layers. The main packages are: *Exception* (handles exceptions), *Factory* (manages the creation of objects) and *Util* (contains utilitarian functions).

2.3. Interface

The graphical interface of the tool is shown in Figure 5.2. The main window consists of four main components:

- *Main menu* (top): The menu allows adding, deleting and editing session components, global components and groups.
- *Graph Viewers panel* (right): A *Graph Viewer* is the editor of a quality graph. It allows displaying a graph, adding/deleting nodes and edges, inserting/deleting groups, changing property values, showing/hiding properties and groups, coloring nodes and edges according to different criteria, zooming and highlighting critical paths. The *Graph Viewers panel* is a container for displaying *Graph Viewers*; it can show several graphs allowing its comparison. In Figure 5.2, the *Graph Viewer* of *Graph1* is active and the *Graph Viewers* of *Graph2* and *Graph3* are iconified.
- *Session Tree panel* (up-left): Shows the components of the current session, structured as a tree, and controls the insertion/deletion of components. The visualization of *Graph Viewers* and the execution of evaluation algorithms are also invoked from this panel. As shown in Figure 5.2, *Session1* is composed of three quality graphs, five data sources, five data targets, two evaluation algorithms and five properties.
- *Group-Property panel* (down-left): When a set of nodes/edges of a quality graph is selected in the *GraphViewer*, their groups and properties are shown in the *Group-Property panel*. The panel also allows inserting/deleting the selected nodes or edges to groups, changing property values, and indicating the properties that are shown/hidden in the *GraphViewer* (for readability purposes). In Figure 5.2, the edge A1-A5 is selected in the *GraphViewer* of Graph1 and consequently its groups and properties are listed in the *Group-Property panel*.

A user manual (in Spanish) can be found in [Ramos+2006].

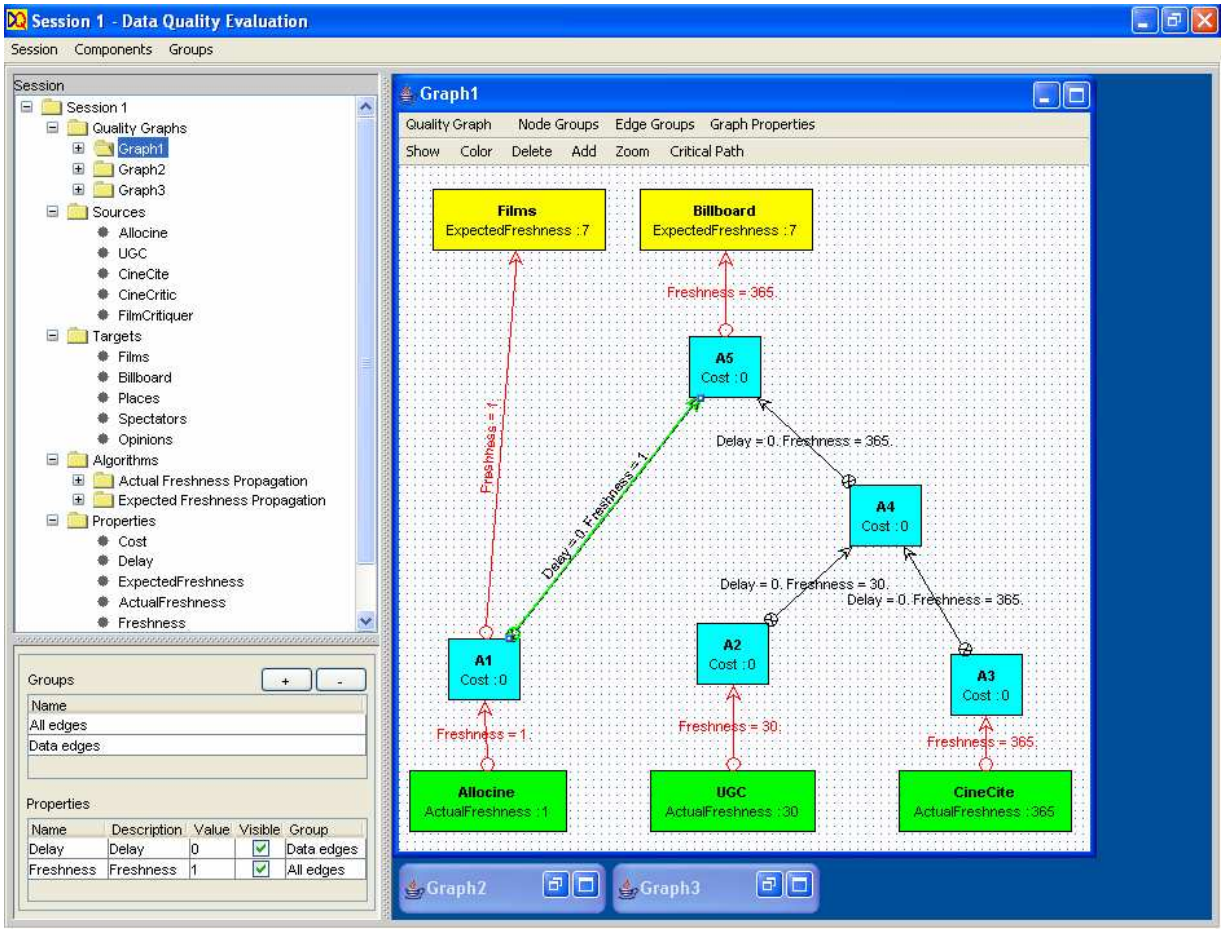

Figure 5.2 – Graphical Interface

2.4. Practical use of the tool

In this section we describe the typical use of the DQE tool for data quality evaluation, not excluding that other sequences of actions can be followed in particular application scenarios and that individual functionalities can be used for other purposes (e.g. visualizing graphs representing other aspects of an application). Therefore, the quality evaluation process using DQE can be abstracted as consisting in two major phases: (i) the personalization of the evaluation algorithms and (ii) their use for evaluating data quality in a DIS. Figure 5.3 illustrates the steps in the quality evaluation process.

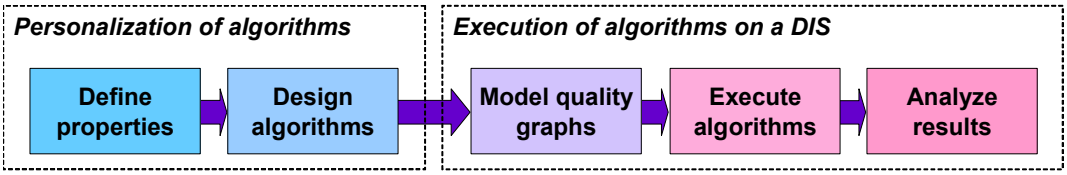

Figure 5.3 – Steps in the quality evaluation process

In the first phase, the user must define (or reuse from the catalogue of properties) the desired quality factors and define (or reuse) the properties that have impact in such quality factors. The tool has a default catalogue of properties that includes all properties mentioned in Chapter 3 and Chapter 4, but the user can add its own properties.

Having defined properties, evaluation algorithms must be implemented, for propagating quality combining such property values. The algorithms for propagating data freshness and data accuracy described in previous chapters, as well as other test algorithms, are included in the default catalogue. In addition, the tool provides an interface and a set of primitive methods for easing the implementation of new algorithms. The user can obtain an evaluation algorithm in three ways:

- Selecting a default propagation algorithm provided by DQE. The property values needed by the algorithm (e.g. source data actual freshness, processing costs and inter-process delays required for the *ActualFreshnessPropagation* algorithm) must be calculated using external methods and stored in the metabase (or manually set in the GraphViewer interface).
- Extending an existing propagation algorithm by overloading the methods for calculating property values. In this case, a new algorithm must be implemented as a Java class extending a default propagation algorithm and overloading the appropriate functions (e.g. overloading the getSourceActualFreshness, getProcessingCost and getInterProcessDelay methods of the *ActualFreshnessPropagation* algorithm, as described in Sub-section 3.5 of Chapter 3).
- Implementing a new algorithm from scratch. To this end, we defined a Java interface (*IAlgorithm*) whose unique method (*execute*) is invoked by DQE for executing the algorithm. The new algorithm must be implemented in a new Java class that implements the *IAlgorithm* interface and the propagation strategy must be implemented in the *execute* method. All the methods of the *QualityGraph* class can be invoked in order to traverse a quality graph with the desired propagation strategy.

In the second phase, the algorithms are used for evaluating the quality of a set of a set of quality graphs. Firstly, a session must be created, selecting the previously defined properties and algorithms, and creating (or reusing) the data sources, data targets and quality graphs. The GraphViewer can be used for creating quality graphs from scratch, editing existing quality graphs or cloning existing quality graphs (e.g. for testing different property values).

Once quality grahs are modeled and property values are associated to their nodes and edges, the evaluation algorithms can be executed. DQE allows executing an algorithm on a selected qualtiy graph or executing an algorithm (in parallel) on all the quality graphs of the session.

After executing algorithms, the GraphViewer can be used again for analyzing evaluation results, for example, highlighting critical paths. Several quality graphs (for example representing alternative implementations of the DIS) can be graphically compared. The evaluation results are also stored in the metabase in order to support decision-making using external tools. In order to support what-if analysis, some property values can be changed (simulating alternative configurations) and the algorithms can be re-executed with the new property values.

2.5. Liberation of versions

The tool first liberation dates from April 2004. The second version, liberated in October 2004, was used for the tests and results that will be discussed in next sections. However, the implementation of the tool was continued, improving persistency and graphical interface, resulting in two additional liberations. Further improvements are scheduled, which are discussed as perspectives in Chapter 6.

History of versions and implementers:

- Version 1 – April 2004: Initial design of the framework; emphasis in the design of the overall architecture and its extensibility (dynamic incorporation of quality properties and evaluation algorithms). Implementation of the main framework components with rapid solutions for evaluation algorithms (basic test algorithms), persistency (via XML files) and graphical interface (ad-hoc components). Developed in a pre-grade project by Fabian Fajardo and Ignacio Crispino, supervised by Verónika Peralta and Raúl Ruggia [Fajardo+2004], presented in the CACIC'2004 conference [Fajardo+2004a].
- Version 2 – October 2004: Incorporation of new functionalities, emphasis in the interoperation with other DIS design tools via a metadata repository. Implementation of freshness evaluation algorithms, incorporation of new display functionalities (as visualization of critical paths), enrichment of the graphical interface and persistence of the framework in the metadata repository. Developed by Verónika Peralta, presented in the BDA'2004 conference [Kostadinov+2004].
- Version 3 – August 2005: Reengineering of data and persistency modules, emphasis in scalability, replacing memory storage by database accesses. Extension of the framework data model, including new features (as groups of nodes and edges) and graphical properties (as colors of nodes and edges), and implementation of the data model in a relational database. Rapid solutions for graphical interface. Developed in a pre-grade project by María José Rouiller, supervised by Verónika Peralta [Rouiller+2005].
- Version 4 – April 2006: Reengineering of the graphical interface, emphasis in easing user interaction. Restructuration of menus and toolbars, incorporation of new display functionalities and implementation of some accuracy evaluation algorithms. Developed in a pre-grade project by Mayra Ramos and Renzo Settimo, supervised by Verónika Peralta, Adriana Marotta and Salvador Tercia [Ramos+2006].

Section 3 describes several application scenarios and the use of the tool for evaluating data freshness and/or data accuracy in them.

3. Applications

In this section we describe some application scenarios where we used the quality evaluation framework. We briefly describe each application scenario, we model it in DQE and we illustrate quality evaluation for it.

3.1. An adaptive system for aiding in the generation of mediation queries

Nowadays, mediation systems are well-known and there exists a great number of implementations. Their main components are: the global schema, mappings between the global schema and the data sources, query rewriting functions and result merging functions. All these components take into account the heterogeneity of data sources which is one of the main problems treated by mediation systems. Other design problems appear at mediator exploitation time. Among these problems we distinguish the definition of the mappings between the global schema and the data sources. Because of a great number of data sources, possibly containing redundant information and different data quality, it is also important to adapt these mappings to user's needs, in particular, in terms of data quality.

In [Kostadinov+2005], we presented an adaptive system for aiding in the generation of *mediation queries* (which represent the mappings between the mediation schema and the data sources). The goals of the system are, on the one hand, to automatically generate mediation queries taking into account data heterogeneity, and on the other hand, to adapt the queries to the user requirements in terms of quality. In this sub-section we describe the system and the use of DQE in the design of mediation applications adapted to user preferences.

The goal of this experiment is twofold: Firstly, we want to validate our approach, instantiating the framework and evaluating data freshness in this concrete application scenario. Secondly, we want to test different evaluation algorithms adapted to different configurations of the system (e.g. materializing data or not).

3.1.1. Description of the application

A mediation system is defined as the integration of several distributed and heterogeneous data sources. The integration is described by means of a global schema, called *mediation schema*, and the mappings relating it to the data sources, called *mediation queries*. Figure 5.4 presents an overview of the architecture of mediation systems.

The generation of mediation queries is one of the most tedious task to be done manually, because of the great number of sources that may be involved (hundreds or thousands) and of the volume of the metadata describing them (source and global schema descriptions, semantic correspondence assertions, etc.). As there may exist several sources providing the same type of data, several mediation queries can define a same mediation object. A key problem is determining the sources (or combination of sources) that provide the results the most adequate to user's needs, according to their thematic and quality preferences. In this context, data quality has a fundamental role in the design and exploitation of mediation applications.

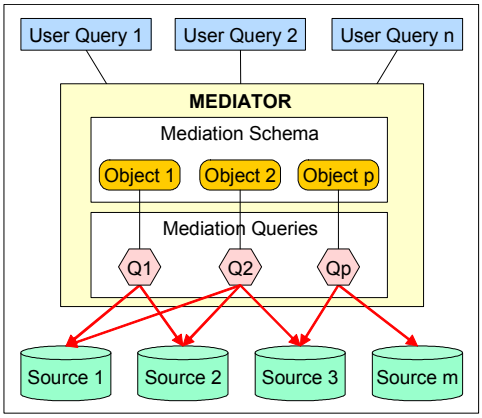

Figure 5.4 – Mediation system architecture

Example 5.1. Consider a mediation system providing information about scientific publications and their authors. The mediation schema is composed of a unique relation:

Publications (**title**, **author**, affiliation, conference, year, editor)*

Also consider four data sources: AuthorBase, ConfList, DBpubs and LP. AuthorBase contains information about authors of scientific publications, ConfList is a catalogue of conferences, DBpubs and LP are lists of published articles. Their schemas are:

- Source AuthorBase : Authors (**author**, address, email, affiliation, nationality)
- Source ConfList : Conferences (**conference**, year, city, country, editor)
- Source DBpubs : Publications (**author**, **title**, conference)
- Source LP : Pubs (**author, title**, conference, year, editor)

In order to calculate the mediation relation the source relations must be combined. In this example, there is not a unique solution. Queries Q_1, Q_2, Q_3 (and their combinations: $Q_1 \cup Q_2$, $Q_1 \cap Q_3$, etc.), allows to obtain the mediation relation:

- Q_1 = LP. Pubs $\bowtie_{author}$ AuthorBase.Authors
- Q_2 = DBpubs.Publications $\bowtie_{author}$ AutorBase.Authors $\bowtie_{conference}$ ConfList.Conferences
- Q_3 = DBpubs.Publications $\bowtie_{author}$ AutorBase.Authors $\bowtie_{auteur,\ titre}$ LP.Pubs

Even if each query allows providing all the attributes of the mediation relation, they produce results with different semantics and different quality. For example, if the source DBpubs is not frequently updated, it can provide data that is less fresh than LP, being less interesting for a researcher searching for new publications. □

* Attributes in bold constitute the key of the relations

The application consists in generating several mediation queries, evaluating the quality of their data and selecting the most appropriate one according to user preferences. A design toolkit provides these functionalities. It is composed of three tools: *mediation query generation*, *user profile management* and *data quality evaluation*. The tools communicate via a metabase server which store all the metadata used and produced by the tools (metadata describing sources, the mediator, mappings, quality measures and user profiles). A definition interface allows users to interact with the tools. Figure 5.5 shows the architecture of the toolkit.

The *mediation query generation* tool [Xue 2006] is responsible for the selection of relevant sources, the detection and resolution of semantic conflicts and the generation of mediation queries. The *user profile management* tool [Kostadinov 2006] is responsible for the definition of user profiles, which include quality expected values. The *data quality evaluation* tool (presented in Section 2) is responsible for the estimation of the quality of data provided by the generated queries and the selection of those satisfying user quality expectations.

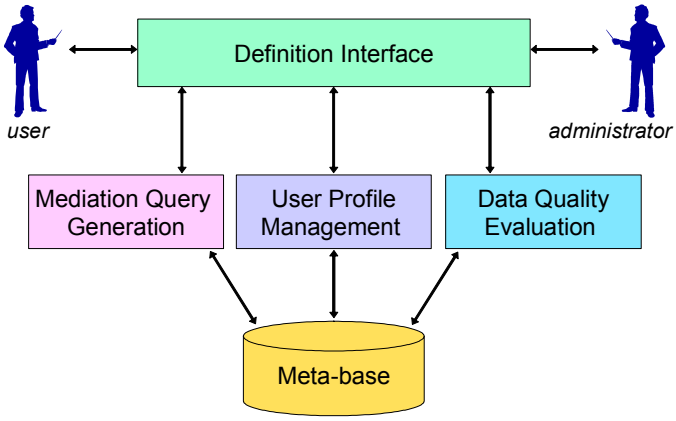

Figure 5.5. Toolkit architecture

In the remaining of this sub-section we discuss quality evaluation features; an overview of the functionalities of the other tools can be found in [Kostadinov+2005].

3.1.2. Modeling of the application in DQE

In this sub-section we illustrate the instantiation of the quality evaluation framework for the application introduced in previous sub-section. Concretely, we model each generated mediation query as a quality graph, allowing the execution of quality evaluation algorithms on the graphs and the comparison among them.

Each query operator is represented as an activity. An additional activity represents the mediator interface for user queries. Figure 5.6 shows a quality graph for the mediation query Q_3 of Example 5.1.

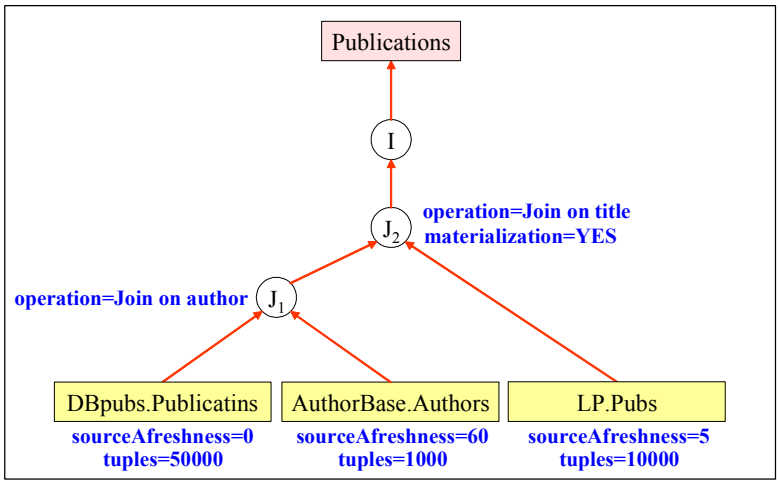

Figure 5.6 – Quality graph for a mediation query

We build two quality graphs for each mediation query, having the same topology but differing in some property values. Concretely, one of the graphs represents a virtual mediation query while the other represents the materialization of the query data. Consequently, synchronization techniques include synchronous and asynchronous pull policies.

Considering that users are interested in measuring both, currency and timeliness, the application scenario is characterized as follows:

- **MED scenario:**
  - Freshness factor: currency and timeliness
  - Nature of data: depends on specific sources (all types of data are possible)
  - Architectural techniques: virtual and materialization techniques
  - Synchronization policies: pull-pull and pull/pull policies

The estimation of property values, taking into account the characteristics of the scenario, is discussed in next sub-section.

3.1.3. Estimation of property values

In order to manipulate concrete freshness expectations, we take as case of study, the domain of scientific publications and their authors introduced in Example 5.1 and we assume that users have very different freshness expectations, ranging from "some days" to "several months" for timeliness and ranging in "some minutes" for currency. Therefore, we need several evaluation algorithms, adapted to different DIS properties and different user expectations.

For timeliness requirements, the "day" is a good unit for measuring data timeliness and properties values ranging in "some hours" or less can be neglected. Query operation costs and inter-process delays among operations are negligible compared with timeliness expectations. However, when materializing data, the inter-process delays with mediator interfaces are relevant. Such delays can be estimated, in the worst case, as the refreshment period. Source data actual timeliness is also relevant. Data timeliness is measured, in the worst case, as the source update period. We assume that external processes are capable of estimating and bounding the time passed since last source update, and that such processes store the calculated value as a source property.

For currency requirements, the "second" should be used for measuring data currency and properties values ranging in "some seconds" should be considered. Query operation costs are relevant. As our purpose was to test the execution of evaluation algorithms and not to compare sophisticated cost models, we simply estimated processing costs based on the number of tuples of source relations. Concretely, the processing cost was computed dividing the number of input tuples (product of numbers of tuples of both input relations, in the case of a join) by the number of tuples that can be processed per second (operation capacity). Note that we need to estimate the number of tuples resulting from each operator (that will be the input for calculating the cost of successor operators). In the case of data materialization (despite materialization is not appropriate for such currency requirements) the inter-process delays with mediator interfaces, estimated in the worst case as the refreshment period, is the predominant delay.

Table 5.1 summarizes the calculation strategies for both factors.

	Data currency	**Data timeliness**
Processing cost (A), one predecessor B	*Tuples* (B) / *Capacity* (A)	Neglect
Processing cost (A), two predecessors B_1 and B_2	*Tuples* (B_1) * *Tuples* (B_2) / *Capacity* (A)	Neglect
Inter-process delay (between operators)	Neglect	Neglect
Inter-process delay (between the last operator A and the interface I)	*Refreshment period* (A)	*Refreshment period* (A)
Source data actual freshness (S), successor A	Neglect	*Update period* (S)

Table 5.1 – Calculation of property values for currency and timeliness factors

When an activity has several predecessors, the maximum input freshness value is taken, i.e. the combination function returns the maximum. (See Chapter 3, Sub-section 3.1 for further details on combination functions).

The following properties were associated to the quality graph:

- RefreshPeriod (last operation node): Represents the difference of time between two refreshments of materialized data. It is defined during mapping generation.
- Capacity (operation node): Represents the quantity of tuples that the node can process per second. It is defined during mapping generation.
- Selectivity (operation node): Represents the percentage of input tuples (product of numbers of tuples of both input relations, in the case of a join) that constitutes the operation result. It is defined during mapping generation.
- Tuples (source and operation nodes): Number of tuples delivered by a node. For source nodes, it is provided by source administrators. For operation nodes, it is calculated from the number of tuples of input nodes and operation selectivity.
- UpdatePeriod (source nodes): Represents the difference of time between two updates at a source. It is provided by source administrators.

Several freshness evaluation algorithms where instantiated, overloading the *getSourceActualFreshness*, *getInterProcessDelay*, *getProcessingCost* and *combineActualFreshness* functions according to the DIS properties discussed previously. Concretely, we provided the following algorithms:

- *TimelinessEvaluation1*: It is the algorithm used in virtual contexts or when user expectations range several months. Processing costs and inter-process delays (including those caused by data materialization) are neglected. Source data actual timeliness is considered.
- *TimelinessEvaluation2*: It is the algorithm used in materialization contexts when user expectations range several days or weeks. Inter-process delays due to data materialization are considered, as well as source data actual timeliness. Processing costs and other inter-process delays are neglected.
- *CurrencyEvaluation1*: It is the algorithm used in virtual contexts. Processing costs are estimated from the number of input tuples and inter-process delays are neglected. Source data actual currency is neglected.
- CurrencyEvaluation2: It is the algorithm used in materialized contexts. Processing costs and inter-process delays among operation nodes are irrelevant compared to refreshment periods, hence, the unique property value that is considered is the inter-process delay caused by data materialization.

The pseudocodes of such functions are detailed in Annex B. Next sub-section describes the experimentations with this application.

3.1.4. Experimentation

As we previously motivated, the experimentation with this applications has two main goals: (i) instantiate the framework and evaluate data freshness in this concrete application scenario in order to validate our evaluation approach, and (ii) test different evaluation algorithms adapted to different DIS properties and user expectations.

To achieve these goals, we implemented several freshness evaluation algorithms, as detailed in previous sub-section, and we simulated different scenarios by changing user preferences. Source property values (Tuples and UpdatePeriod) were manually set during test configuration. DIS property values (RefreshPeriod, Capacity and Selectivity) were randomly generated during query generation. All these properties, as well as the quality graphs representing mediation queries, were read from the metabase.

We executed the evaluation algorithms and used the graphical functionalities of DQE for analyzing evaluation results. Two main functionalities were particularly used: the coloring of graphs not achieving freshness expectations and the coloring of critical paths. This allowed simulating DIS reengineering analysis. Furthermore, some property values were manually changed during the simulation (using the Group/Property panel of DQE), for example, the refreshment frequency or the capacity of a node. Then, the evaluation algorithms were re-executed with the new property values, allowing the simulation of some basic kinds of *what-if* analysis.

The evaluation results, i.e. labels of the quality graphs, were persisted in the Metabase and thereafter used by the *User Profile Manager* in order to select the mediation query that better adapts to user preferences. The selection principle was very simple: mediation queries were ordered according to data freshness and other criteria, and the one providing the best quality (in a multi-criteria aggregation) was selected.

In order to provide other quality criteria, we implemented simple quality evaluation algorithms for computing *response time* and *confidence* estimations. Response time was computed adding the processing cost property calculated for data freshness evaluation. Confidence was computed as an average of the source confidence values, set as properties of source nodes. Despite the simplicity of the test, this allowed to validate the possibility of extending the framework (and the tool) for the evaluation of other quality factors.

As results of our experimentations we obtained some conclusions. Firstly, the application of the instantiation method to a given scenario is straight forward. In addition, we found that the knowledge about many properties and its estimation can be reused for other scenarios. The implementation of the overloaded functions is also very simple and its integration to the framework is straight forward. Finally, the graphical functionalities of the tool allowed an easy interpretation of evaluation results, especially, the visualization of critical paths is very useful for targeting enforcement analysis.

3.2. Evaluating data freshness in a web warehousing application

Web warehousing (WW) systems extract and integrate data from a set of conceptually-related web pages and store it in a Data Warehouse in order to allow decision making. WW extraction processes have to solve many problems related to the heterogeneity of web pages and the autonomy of web sources, including finding new relevant pages, detecting changes in the pages, accessing pages with different formats and transforming data to a common format. As many sources can provide the same information, extracted data must be reconciled in order to provide consistent and not duplicated information.

In this sub-section we study data freshness evaluation in the context of a WW application. Concretely, we instantiate the freshness evaluation algorithm to a WW scenario and we use the evaluation results in the integration process in order to return the freshest data to the user.

The main goal of this experiment is the practical use of our approach in a real application. This implies the modeling of several types of processes with different synchronization policies and different assessment methods for property values. The challenge is to show that our framework allows the easy representation of the application and that the freshness evaluation algorithm can be easily instantiated for this scenario.

3.2.1. Description of the application

In this sub-section we describe the Web Warehousing architecture proposed in [Marotta+2001], which is based on two types of modules: *wrappers* and *mediators*. The goal of a wrapper is to access a source, extract the relevant data, and present such data in a specified format. The role of a mediator is to merge data produced by different wrappers or mediators. Figure 5.7 shows an overview of the Web Warehousing architecture, starting with data extraction from Web documents and finishing with user interfaces or applications querying the system.

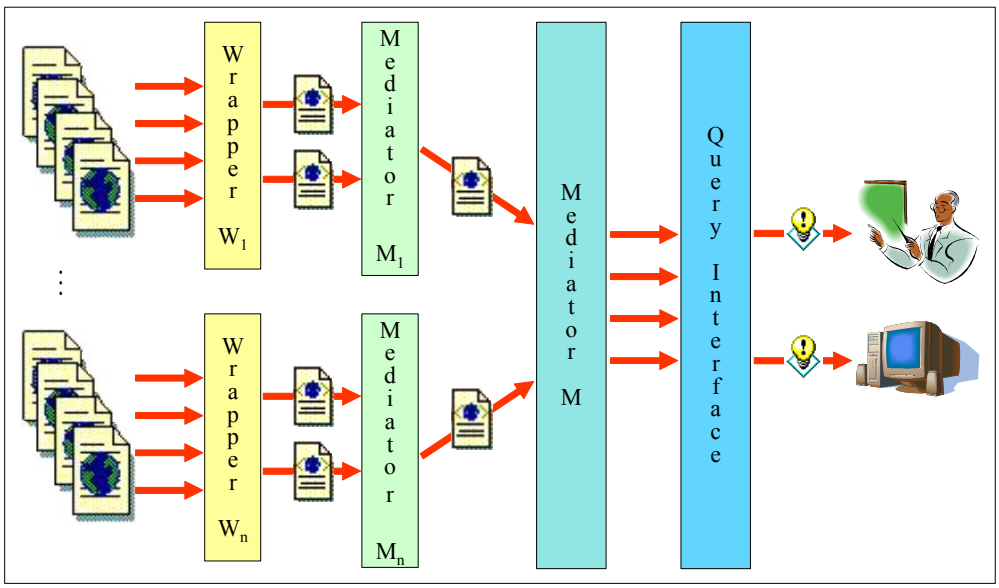

Figure 5.7 – Overview of the Web Warehousing architecture

The set of Web pages is grouped according to the information to be extracted from them. For each group of pages, a request schema and an associated wrapper are defined. The wrapper extracts the data in the pages and structures it according to the request schema. The output of each wrapper is a set of identical schemas populated with the corresponding data of each page. The wrappers also detect changes in Web pages.

A global integrated schema is also defined, which represents a unified view of the data manipulated by the system. As a direct consequence, data integration follows the local as view approach, where each data source is defined as a view of the global schema. The proposed architecture contains two kinds of mediators: (i) those integrating the results returned by each wrapper (called M_i) which only perform instance mappings since input schemas are identical, and (ii) a global mediator integrating the data returned by the M_i mediators (called M) which performs schema and instance mappings.

We briefly describe each module; a larger description can be found in [Giaudrone+2005] [Marotta+2001].

- *Wrappers*: Information search is oriented by a request schema (expressed in an XML schema). The information request is build using all concepts of the request schema, incorporating synonyms of the concepts (e.g "movie" and "film") and instances of the concepts (e.g. "movie" and "Harry Potter") obtained from a domain ontology. It is used for identifying relevant Web pages and retrieving data from them. The retrieval algorithm is based on page structure; it evaluates rules for each element of the information request in order to locate it in the page. The result is an XML document for each relevant page, formatted according to the request schema.
- *M_i mediators*: An M_i mediator integrates the XML documents returned by wrappers, solving possible data conflicts. When a same element is found in several input pages, conflicting attributes are solved using a source-trust policy (the value from the source with the greater trust is chosen), non-conflicting attributes are taken from its source. The integration process can be extended to handle other conflict-resolution policies. The mediator also uses the domain ontology for unifying terms (e.g. "United States", "EE.UU" and "USA") and performs some basic cleaning for refusing invalid data elements (e.g. those containing null values for mandatory attributes). The result is an XML document that contains all extracted information for a request schema.
- *M mediator*: The M mediator integrates the data produced by M_i mediators, also solving structural conflicts. The final result is a relational DW. The DW schema is designed by applying a set of high-level transformations to the request schemas [Marotta 2000]. Each transformation takes as input a set of relations and produces as output a set of relations, corresponding to high-level operations such as aggregations, denormalizations, calculations, etc. Data conflicts are solved following the approach of [Calvanesse+1999], which declaratively specifies suitable matching and reconciliation operations.

Regarding synchronization policies, some sources have the capability of announcing changes to wrappers (push policy). After being notified of a change in a page and having verified that it is a significant change (not only a format change) the wrapper executes. For the other sources, the wrappers periodically compare pages with their previously recorded versions (pull policy). Mediators follow push policies, which means that they execute when a previous module produced a new output (an XML document, input for the mediator). It is possible that changes are not reflected in the mediator output, for example, if the new data was duplicated in other source, or if it is conflicting with data coming from a source with higher priority. All modules materialize data, either in XML documents or in a relational database. User queries follow pull policies. Other application scenarios with different synchronization policies where studied in [Peralta 2006].

The following case of study (taken from [Giaudrone+2005]) illustrates the approach along the sub-section:

> **Example 5.2.** Consider a web-warehousing application that extracts web data about cinema. The application has two wrappers, the one extracting information about movies (title, original title, country, year, duration, genre, etc.) and the other extracting information about actors (name, nationality, place of birth, etc.). Figure 5.8 shows an example of a web page with information about a movie, showing some attributes of the request schema that where recognized by the wrapper in the page text. Detailed information about the request schemas, as well as the ontologies describing such concepts, can be found in [Giaudrone+2005].
>
> Two mediators: M_1 and M_2 take as input the XML documents with the extracted information and produce two XML documents integrating extracted data. Mediator M transforms such documents to the relational model, integrates and transforms data, populating five materialized views that conform the DW. Users access the DW data through view interfaces, which allows queries on *producers*, *movies*, *plays, statistics on plays* and *actors*. □

Attribute	Description
Título	Title (Spanish)
Título original	Original title (original language)
Paises	Production countries
Año	Opening year
Duración	Duration in minutes
Genero	Artistic genre
Productora	Production company
Audiencia	Number of tickets sold
Idioma	Original language
Recaudación	Total takings
Actores	Names of the main actors
Directores	Names of the directors
Productores	Names of the producers

Figure 5.8 – Extracting information about an actor: (left) web page, (right) request schema

The following sub-section shows how this case of study is modeled in DQE.

3.2.2. Modeling of the application in DQE

In this sub-section we illustrate the instantiation of the quality evaluation framework for the case of study introduced in previous sub-section. Concretely, we model quality graphs, describing the representation of the activities and their interaction. We consider a simplified version of the case of study, accessing to a small number of web pages ($\{mov_1, mov_2, mov_3\}$ for movies and $\{act_1, act_2, act_3, act_4\}$ for actors).

The quality graph modeling the application is shown in Figure 5.9. Wrappers execute for each page, generating an XML document with the extracted information. We represented several instances of wrappers (activities w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, w_{23} and w_{24}) to explicitly show the data flow, however, there is a unique process that is invocated for several pages. Mediators M_1 and M_2 take as input the XML documents with the extracted information and produce another XML document integrating all extracted data; they are represented by activities m_1 and m_2. Mediator M integrates information produced by mediators M_1 and M_2. Input data is transformed to the

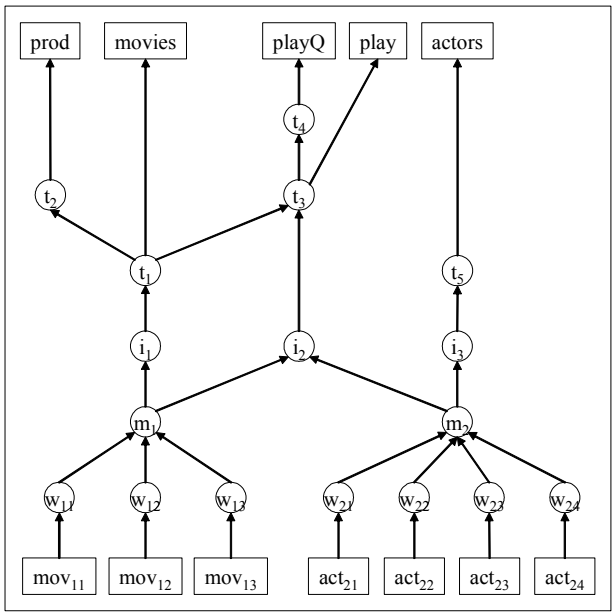

Figure 5.9 – Quality graph for the web warehousing application

relational model and stored in two ODS relations (activities i_1 and i_3). The DW is populated by a set of processes that transform ODS data, represented by activities t_1, t_2, t_3, t_4 and t_5. Users access the DW data through the view interfaces v_1, v_2, v_3, v_4 and v_5. Control flow coincides with data flow. The data types of the data produced by each node (e.g. XML documents, temporal relations, materialized views) are labels of the quality graph.

Let's analyze some characteristics of the scenario:

- Users are interested in seeing "recent" information about films and actors (timeliness). However, system administrators (who monitor change detection and view refreshment) need currency measures.
- The nature of data is different in the various sources. Actors' information (name, nationality, etc.) is quite stable, some movie information (title, country, genre, etc.) has a relatively long-term change frequency but some attributes (audience, takings, etc.) frequently change.
- Concerning DIS implementation, the application materializes data. Furthermore, all activities materialize data, either in XML files or in a relational database. Activities are simple automatic extraction, reconciliation and aggregation processes.
- There are different synchronization policies: synchronous push and asynchronous pull between sources and wrappers, synchronous pull between view interfaces and targets, asynchronous pull between transformations and view interfaces, and asynchronous push among the other activities.

The scenario is characterized as follows:

- **WW scenario:**
  - Freshness factor: currency and timeliness
  - Nature of data: stable, long-term-changing and frequently-changing data
  - Architectural techniques: materialization techniques
  - Synchronization policies: push-push, pull-pull and pull/pull policies

The characteristics of the scenario are taken into account to determine the relevant properties and estimate them, which is discussed in next sub-section.

3.2.3. Estimation of property values

Users expect timeliness values of "two weeks" for movies, play, statistics and producers' information and of "six months" for actors' information. System administrators tolerate currency values of "a week" for all user interfaces. With such freshness requirements, the "day" is a good unit for measuring freshness and properties values. Properties with values ranging in "some minutes" or less can be neglected. Activities processing costs are negligible compared with freshness expectations. However, due to data materialization, inter-process delays are relevant. In addition, as many sources may rarely update data, source data actual timeliness is also relevant.

We can now estimate the values of relevant properties, i.e. inter-process delays and source data actual freshness.

Data currency is measured as the time passed since data extraction, so it is zero at extraction time (source data actual currency). Data timeliness is measured as the time passed since page update[*]. The estimation of update time depends on the synchronization policy between sources and wrappers. For wrappers with push policy the notification time is used. Wrappers with push policy extract data immediately after being notified of a change, so source data actual timeliness is negligible for them. For wrappers with pull policy, source data actual timeliness is more difficult to estimate. When a change is detected (comparing with the previous version) the time passed between two consecutive pulls (pull period) is a worst case estimation. When no change is detected, the time passed since the last change detection should be added. More precise estimations may be obtained from knowledge about sources (e.g. if we know that a source is updated every Monday) or using statistical techniques (e.g. frequently accessing a source during some periods to determine source update behaviors). Some sources allow the access to file system metadata (as file modification time) or include the "last update time" inside the page text. Our tests have shown that none of these indicators are reliable enough. The former depends on system configuration (regional settings and correct configuration of local time) and the latter is not always updated when modifying a page.

[*] Depending on user needs, display changes can be ignored.

Due to push policies, activities having a unique predecessor (which is the case of i_1, i_2, t_1, t_2, t_4 and t_5) always execute after their predecessor, so there is no inter-process delay among them. However, an activity having several predecessors (which is the case of m_1, m_2 and t_3) executes when an input activity produces new data but may read data materialized several days ago by the other input activities. The inter-process delay among them is measured as the difference of time between their executions. Keeping statistics of execution dates the precise calculation is straightforward. Average values and upper bounds can be also obtained from statistics or can be deduced from source update policies*. Note that as some modules may not produce output data (e.g. if the changes are not relevant or where provided by other sources), successor activities do not need to execute because they would produce the same output too. Regarding data, the output of successor modules will be identical to the existing one, nevertheless, regarding data quality, the output will be fresher because it is built with more recent data (the page was updated). So, the execution date of all successor activities is updated even if they do not execute. The time passed between the materialization of data and the moment a user poses a query, determines the inter-process delay between transformations and view interfaces. Precise values, average values and upper bounds can be also obtained from statistics of execution dates.

Table 5.2 summarizes the calculation strategies.

	Precise value	**Average case**	**Worst case**
Processing cost (A)	Neglect	Neglect	Neglect
Inter-process delay (A,B), B in {i_1, i_2, t_1, t_2, t_4, t_5}	Neglect	Neglect	Neglect
Inter-process delay (A,B), B in {m_1, m_2, t_3, v_1, v_2, v_3, v_4, v_5}	*Last execution time* (B) – *Last execution time*(A)	Average in statistics of: *Execution time* (B) – *Execution time* (A)	Maximum in statistics of: *Execution time* (B) – *Execution time* (A)
Source data actual currency (S)	Neglect	Neglect	Neglect
Source data actual timeliness (S), push	Neglect	Neglect	Neglect
Source data actual timeliness (S), periodic pull, wrapper W	*Last execution time* (W) – *Last change detection* + *Pull period* (W)	*Pull period* (W) + Average in statistics of: *Execution time* (W) – *Change detection time*	*Pull period* (W) + Maximum in statistics of: *Execution time* (W) – *Change detection time*

Table 5.2 – Calculation of property values with different types of estimation

Remember that when an activity has several predecessors, the freshness of data coming from them is combined, consolidating an input freshness value for the activity and that the combination strategy depends on the activity semantics. In our case of study we take into account data volatility (movie information is more volatile than actors' information); the strategy consists of ignoring input values of sources providing more stable information. For example, t_3 will return the input value from t_1. The nature of data dimension is used to define this strategy. When input activities have the same data volatility (which is the case of wrappers), the combination function performs a weighted average, where weights are data volumes (number of movies/actors). Table 5.3 summarizes the calculation strategies.

	Precise value	**Average case**	**Worst case**
When different data volatility	Input value of most volatile input		
When equal data volatility	Average of input values weighted with *Data volume*	Average in statistics of: average of input values weighted with *Data volume*	Maximum (input values)

Table 5.3 – Calculation of combination functions with different types of estimation

In next sub-section we discuss the instantiation of freshness evaluation algorithms taking into account these properties.

* Despite source autonomy, in some application domains it is common to have information about source update policies. For example, in the cinema domain, most timetables pages are updated weekly (e.g. at Wednesday) because cinema schedules generally change weekly.

3.2.4. Data freshness evaluation

The freshness evaluation algorithm is instantiated overloading the *getSourceActualFreshness*, *getInterProcessDelay*, *getProcessingCost* and *combineActualFreshness* functions according to the DIS properties that are most relevant for the scenario. Pseudocodes of such functions are detailed in Annex 2.

The following properties were associated to the quality graph:

- AnnounceChanges: Represent the capability of the source to announce changes to the corresponding wrapper. It is defined by DIS administrators and set as propagate value (at DIS design time).
- PullPeriod (wrapper nodes): Represents the difference of time between two data extractions. It is defined by DIS administrators and set as propagate value (at DIS design time).
- DataVolatility (data edge): Represents the volatility of data flowing in the edge. Expert users provide the volatility values for wrappers, which are replicated to data edges and set as propagate values (at DIS design time).

The following logs were used to register more volatile properties:

- Execution-ChangeDetection (source, wrapper): At each execution, a wrapper registers two values: (i) its execution time and (ii) the previous change detection time. The log provides statistics of the difference between such values, i.e. the last, average and maximum entry.
- Execution-Execution (activity, activity): At each execution, a mediator task or view interface registers two values for each predecessor module: (i) the execution time of the predecessor (obtained from the output of the module or a previous entry of this log) and (ii) its execution time. The log provides statistics of the difference between such values, i.e. the last, average and maximum entry.
- Execution-ChangeDetection (source, wrapper): At each execution, a wrapper or mediator task registers one value: the number of produced tuples/elements. The log provides statistics on such values, i.e. the last, average and maximum entry.

Statistics on execution times and change detection times where recorded by wrappers and mediators [Vila+2006]. Statistics on user queries have been recorded by triggers on view interfaces (we simulated random accesses in order to test the proposal).

Next sub-section describes the experimentations with this application.

3.2.5. Experimentation

In order to test the practical use of our approach in a real application, we modeled the WW application as a quality graph and we analyzed its properties. The main difficulty resided in the modeling of different update propagation policies, which caused different ways of determining inter-process delays. The assessment of source data freshness was an additional challenge. Several property values (announcement policies, pull periods and data volatility) were determined at design time and set as labels of the quality graph. However, other property values (amounts of time among execution of activities) are more volatile and needed fresh statistical estimations in order to assess property values. We measured such property values from logs recorded by wrappers and mediators. Such modules were explicitly modified in order to record such statistics; details on the implementation of logs can be found in [Vila+2006].

A first conclusion of our experimentation is that, even the DIS and its relevant properties can be easily modeled in DQE, the assessment of source data quality and DIS property values can be a tedious task and may imply the development of specialized routines. In this thesis we do not deal with source quality and property assessment, but we found that the analysis of assessment techniques can be an interesting line for future works. This topic is discussed in Chapter 6.

Concerning the implementation and execution of evaluation algorithms, we confirm the conclusions obtained in previous experiment, i.e. the application of the instantiation method to a given scenario is direct, the implementation of the overloaded functions is very simple and its integration to the framework is straight forward.

An indirect result of the experimentation was the use of quality values to improve the data integration process. Concretely, the M_i mediators where modified in order to take as input the freshness of data extracted by wrappers. Data quality is introduced in the integration process in two ways:

- *Data filtering*: The sources do not achieving a certain quality level are discarded. To this end, each mediator has associated a quality bound (a freshness threshold) and does not process inputs (XML documents corresponding to Web pages) which freshness value is greater than the threshold.
- *Reconciliation policy*: The source with the higher quality is selected when there are conflicts. To this end, a new reconciliation policy was defined for the WW application, based on the freshness of conflictive data, i.e. the data coming from the freshest input (XML document corresponding to a Web page) is chosen.

Clearly, the new implementations of mediators produce fresher data than their previous versions. However, as data freshness is not the unique criterion that should be taken into account in data reconciliation, the real improvement has not been tested. We will discuss it as perspective.

3.3. Evaluating data accuracy in a data warehousing application

The loading and refreshment of a Data Warehouse (DW) involves several types of activities, ranging from the extraction of data from several sources, the transformation of this data and its loading into the DW. Such processes is generally know as ETL (*Extraction, Transformation and Loading*) processes. The transformations applied to extracted data consist, basically, of data cleaning and formatting routines. Data cleaning is necessary to assure the quality of DW data. It involves, among others, the correction of errors, the elimination of redundancy and the resolution of inconsistencies, as well as the achievement of the business rules. Formatting serves to structure data according to DW requirements. It includes the adjustment of data to DW schema, the changing of data format and the aggregation of data.

In this section we analyze the ETL processes of a concrete DW application, which manages information about students of a university [Etcheverry+2005].

The goal of this experiment is threefold. Firstly, we want to model a big complex application in DQE, having many activities and varied properties. Secondly, we intend to experience with different accuracy metrics (for different data types and error types) and with varied measurement techniques. Finally, we want to validate the accuracy evaluation approach in a real application.

3.3.1. Description of the application

We limited our analysis to the ETL processes of a data mart of the university DW. The data mart consists of two multidimensional fact tables and eleven dimensions, which describe university students, the courses they followed, the exams they took and their marks.

ETL processes where studied at three levels of abstraction: The highest level provides a macro vision of ETL packages and their precedence relationships. There are 3 packages, one for loading dimensions and one for loading each fact table. The medium level describes the dataflow inside each module, identifying ETL routines (transactions). The lowest level describes the sequence of operations that implement each ETL routine. Each ETL module contains between 5 and 20 routines, each one containing up to 10 operations. Figure 5.10 illustrates the routines that compose the loading of one of the fact tables (medium level). A complete description of ETL modules, at the three abstraction levels, can be found in [Etcheverry+2005].

The modeling of the application in DQE is straight forward. We defined a hierarchy of quality graphs, with three levels, representing activities at the three abstraction levels. The highest-level graph allows visualizing the whole process; activity nodes represent ETL packages. The medium-level graph represents the dataflow among ETL routines and the lowest-level graph shows ETL operations. Source nodes represent source relations and target nodes represent dimension and fact tables. See Sub-section 4.1.1 of Chapter 3 for details on modeling quality graphs at different abstraction levels.

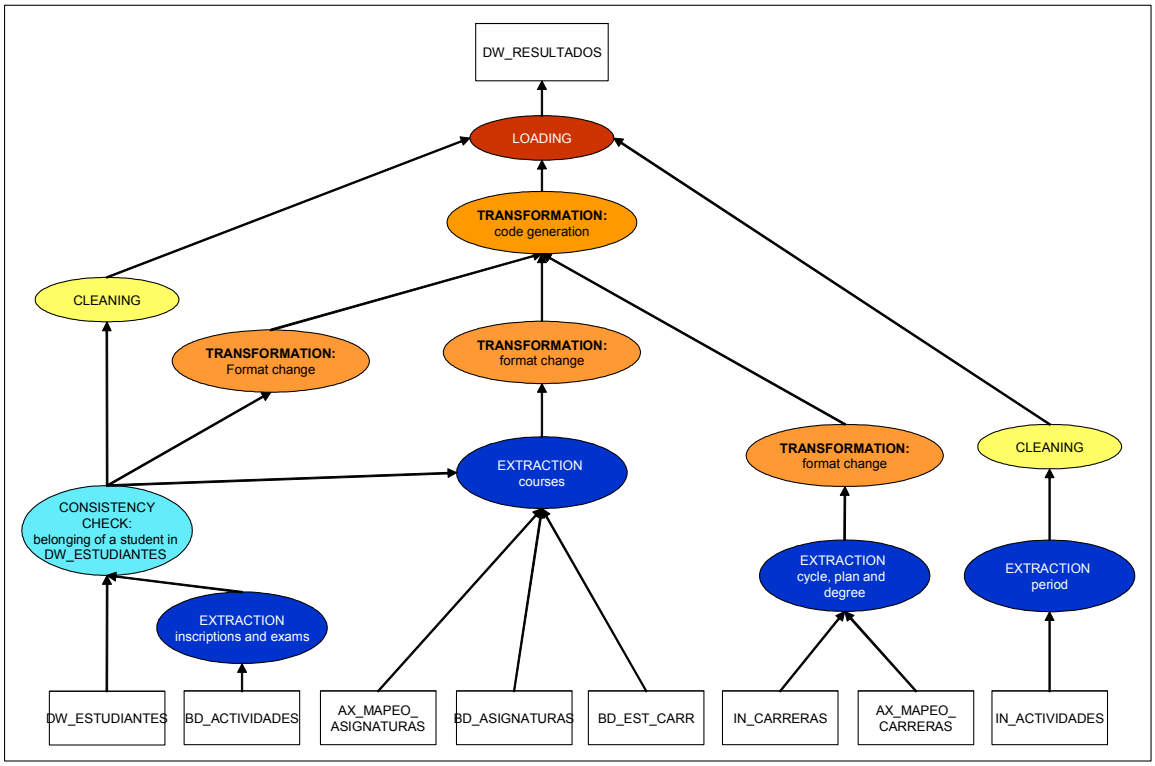

Figure 5.10 – Routines of an ETL package: extraction, consistency check, transformation, cleaning, and loading.

3.3.2. Accuracy assessment

In order to measure accuracy of source data, we experienced several metrics and measurement functions. The implementation of the assessment techniques was carried out by two master students: Lorena Etcheverry and Salvador Tercia [Etcheverry+2006], the former having worked in the development of the DW application some years ago. They implemented specialized techniques for detecting and measuring four types of errors*, which are summarized in Table 5.4.

The assessment of semantic correctness was simulated in order to avoid comparisons with real-world. Source relations were polluted in order to represent semantic errors. Copies of source relations (before pollution) were used as referential tables representing real world. The assessment method (*CHECK_REF* function) consisted in the comparison of source data with referential data. This pollution strategy allowed the implementation of automatic assessment methods.

An assessment function (*CHECK_RULE*) was implemented for measuring syntactic errors. The function verifies if data satisfies a given format, given by a range or a grammar. Several attributes already contained errors but further errors were simulated either polluting source data or defining restricted formats. An example of the latter was the definition of a grammar, consisting of 7 digits, for telephone numbers. As in Uruguay home telephones have 7 digits and mobile telephones have 9 digits, mobile telephones are detected as having an illegal format.

We detected several cases of imprecision in source data. For example, most instances of *city* attribute (DW_ESTUDIANTES table) correspond to cities, but we also found country names and continent names. We defined a referential table containing the instances of the *city* attribute and assigning them a level of precision. Specifically, precision is 1 for city names and takes inferior values for country and continent names. The assessment function (*CHECK_LEVEL*) obtains the precision measure from the referential table. Implementation details can be found in [Etcheverry+2006].

* The work also included the assessment of data consistency, considering further types of errors, including the presence of null values for mandatory attributes and the violation of attribute dependencies, uniqueness constraints and referential integrity constraints. As the analysis of data consistency is out of the scope of this thesis, we do not provide details on its measurement.

Accuracy factor	Error description	Assessment method
Semantic correctness	Values that not correspond to a real-world entities.	*CHECK_REF*: Compares data against a referential table representing real-world.
Syntactic correctness	Out or range values	*CHECK_RULE*: Evaluates if data satisfies a range rule (expressed as a numeric interval)
	Illegal format	*CHECK_RULE*: Evaluates if data satisfies a format rule (expressed by a grammar)
Precision	Insufficient precision	*CHECK_LEVEL*: Compares data precision against a referential table

Table 5.4 – Typical errors and their corresponding assessment functions

3.3.3. Experimentation

In order to test our accuracy evaluation algorithm we defined an application scenario inspired on the DW loading processes. We evaluate accuracy of data loaded from DW sources, i.e. data stored in the DW. As our evaluation approach does not considers data materialization and only manages JSP queries, ETL processes were simplified to JSP queries, ignoring data cleaning, consistency checks and complex transformations. The simplified loading queries allowed the validation of our approach with a representative set of queries.

Source relations were partitioned based on accuracy measures (using the assessment functions explained in previous sub-section). Several partitioning criteria were used. In some cases we used knowledge about error distribution (obtained executing test queries and keeping statistics); in other cases we designed pollution methods in order to obtain certain error distribution (e.g. with defined a set of areas and we polluted them with different error ratios). It was not necessary, for our tests, to use Rakov's partitioning algorithm. The loading queries were rewritten in terms of partitions. The selectivity of each rewriting was estimated using simple statistics.

This test allowed the validation of our measurement approach with real data. We experimented with different accuracy metrics and assessment functions as well as with different partitioning criteria. We can affirm that the approach is viable and obtained accuracy values may approximate the actual values obtained assessing accuracy of exact query result. We plan to make additional tests in order to measure the precision of the obtained accuracy values. In Chapter 4 we showed that the evaluation approach depends on the techniques used for measuring accuracy of source relations, partitioning them and estimating query selectivity. We aim to quantify such influence by testing different techniques and their impact in the partitioning of query results. By performing such tests, we can compare results with those obtained applying the Naumann's approach [Naumann+1999] (which considers uniform distribution of errors) and the Rakov's approach [Rakov 1998] (which is build knowing the exact query result). Both approaches were described in Sub-section 3.1 of Chapter 4. We defer this test to near future.

As future work, we hope to extend the approach to consider other types of activities, specially, those correcting errors (e.g. data cleaning and format standardization routines). This DW application can be taken as a case of study for proposing evaluation techniques and algorithms for this type of application scenarios. The measurement approach proposed in Chapter 4 is a first step towards the definition of general evaluation methods that might be instantiated to several types of application scenarios. This topic is discussed as perspective in Chapter 6.

Another result of this experiment, was the use of DQE for modeling a big and complex DIS, with many activities and varied properties (materialization, complex transformations, etc.).

4. Evaluation of Performance and Limitations of the DQE Tool

In this section we present the results of performance and limitations tests performed with the DQE tool.

In order to evaluate performance and limitations, we generated some test data sets (consisting in a set of quality graphs, data sources, data targets and property values) and we executed an evaluation algorithm (the *ActualFreshnessPropagation* algorithm) on each graph. The second version of DQE was used for the tests, so quality graphs are loaded from the metabase and stored in memory. Thus, one of the objectives of the tests was

to determine the limitations of the tool, i.e. the number of quality graphs that can be loaded in a session being the input for the execution of quality evaluation algorithms. We made three types of tests:

- *Application limits*: The test consisted in generating large data sets in order to know the number of graphs (with different topologies) supported concurrently by the tool.
- *Quality evaluation performance*: The test consisted in measuring the time it takes the tool to evaluate data quality on a set of graphs, i.e. executing the evaluation algorithm on the graphs.
- *Loading performance*: The test consisted in measuring the time it takes the tool to communicate with the metabase for loading the set of graphs and their associated metadata.

The obtained results allow affirming that the tool can be used for large applications (modeling hundreds of graphs with hundreds of nodes each). Scalability (using the last DQE version) is discussed as perspective in Chapter 6. The following sub-sections discuss the generation of test data sets, describe the tests and present the results.

4.1. Generation of test data sets

In this section we describe the data sets used for the various tests, the process used for generating them and the results of the generation.

We generated three types of test data sets:

- *Small data sets*: Fifty data sets where generated varying the number of graphs (from 10 to 100) and the number of nodes of each graph from (10 to 50).
- *Bulk data sets*: Fifty data sets where generated varying the number of graphs (from 100 to 5000); graphs are small (15 nodes).
- *Big data sets*: A hundred data sets where generated varying the number of graphs (from 100 to 1000) and the number of nodes of each graph from (100 to 1000).

The main purpose of generating small data sets is validating the correctness of the generator (checking if generated graphs are well-formed, with varied topologies and adequate property values, allowing the correct execution of the evaluation algorithm). In addition, as most application scenarios are represented with small graphs, we also are interested in evaluating the tool performance. The purpose of generating bulk and big data sets is to know the maximum number of small/big graphs supported in memory and the performance of the quality evaluation algorithms with such data sets.

Next sub-section describes the generation process.

4.1.1. Generator of data sets

We implemented a generator of data sets, which works in three phases: (i) generates a set of sources and a set of targets, (ii) generates a set of quality graphs, and (iii) associates property values to the nodes and edges of the graphs.

The first phase is straightforward. Sources and targets are sequentially named (e.g. S1 and T1).

In the second phase, a set of graphs is also generated, with sequential names (e.g. graph1). Nodes and edges are generated in six steps, as illustrated in Figure 5.11 (all edges are mixed data edges, representing data and control flow). The generation steps are the following:

1. A set of source nodes is randomly selected from the source set and a set of target nodes is randomly selected from the target set. In Figure 5.11a, three sources and two targets are selected.
2. A set of activity nodes is generated, sequentially named (e.g. A1). In Figure 5.11b, six activities are generated.
3. A set of edges between source and activity nodes is generated in the following way: for each source node, an activity node is randomly selected (a different activity for each source node) and an edge between them is added. In Figure 5.11c, activities A6, A2 and A1 are selected.
4. A set of edges between activity nodes is generated in order to link each isolated activity to the ones already connected (initially, those linked to source nodes). To do so, a set of activity nodes are randomly

selected from those already connected (which will be the predecessors) and an edge between each predecessor and the isolated activity is added. In Figure 5.11d, A6 and A2 are selected as predecessors for A3; then A5 is linked to A6 and A4 is linked to A3.

5. A set of edges between activity and target nodes is generated, in the following way: for each target node, an activity node is randomly selected (a different activity for each source node) and an edge between them is added. Activities with no successors have priority to be selected. In Figure 5.11e, activities A5 and A4 are selected.
6. If there are activities with no successors, a set of edges is generated for connecting them to other nodes. In order to avoid cycles, candidate nodes are those that are predecessors of some target node. In Figure 5.11f, an edge between A1 and A4 is added.

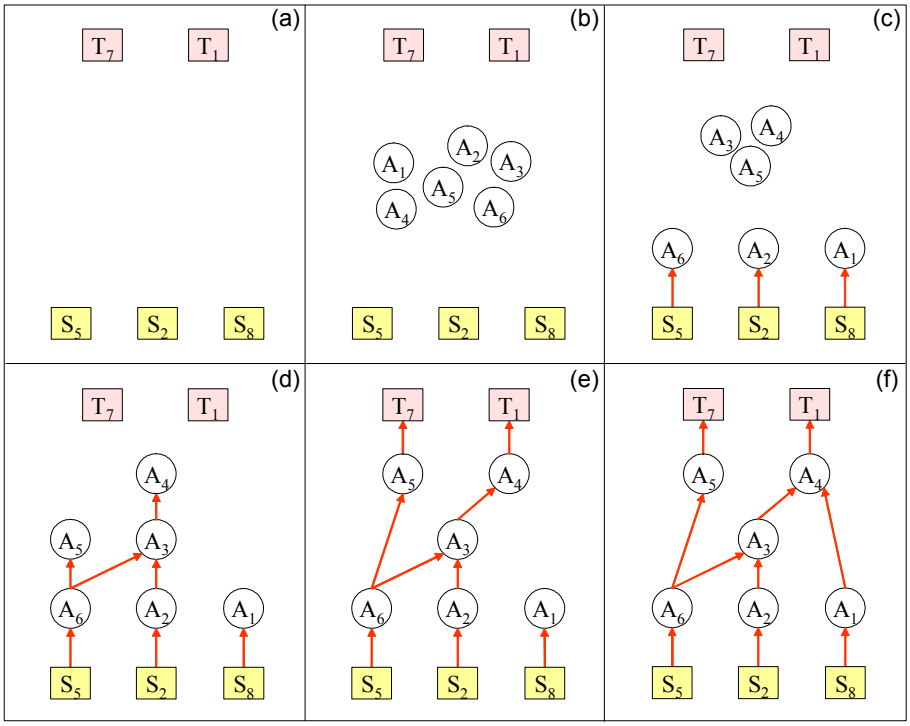

Figure 5.11 – Generation of graphs

In the third phase, a set of property values are associated to nodes and edges, in the following way:

- A source data actual freshness value is associated to each source, and then, to all source nodes (of the graphs) corresponding to the source.
- A target data expected freshness value is associated to each target, and then, to all target nodes (of the graphs) corresponding to the target.
- A processing cost value is associated to each activity node of the graphs.
- An inter-process delay value is associated to each edge of the graphs.

Table 5.5 lists the generation parameters and their default values. Four categories of parameters are defined: session components, number of graph nodes, input degree of activity nodes and property values. All these parameters are written in a configuration file.

Table 5.6 summarizes the parameters used for generating the three types of data sets.

The generator was implemented in Java (JDK 1.4) and stores the generated data set in an Oracle® database (Oracle 10g Enterprise Edition Release 10.1.0.2.0). In next sub-section we discuss a preliminary test performed for verifying that the generated graphs are well-formed and allow the execution of quality evaluation algorithms.

Parameter	Description	Default
Session components. Sources, targets and quality graphs are generated; properties and algorithms are fixed.		
S	*Number of sources* in the session (integer number)	
T	*Number of targets* in the session (integer number)	
G	*Number of graphs* in the session (integer number)	
Number of graph nodes. As nodes are of three types (source nodes, activity nodes and target nodes), three parameters are defined, one for each type. In order to generate heterogeneous graphs, the number of nodes is randomly set, given the intervals (minimum and maximum values) for the random selection. Some integrity rules assures that minimum values are appropriate (smaller than maximum values and, in the case of activities, bigger than the number of sources and targets randomly generated). Parameters are:		
$\mathbf{N_s}$	*Maximum number of source nodes* of the graphs (integer number)	
$\mathbf{n_s}$	*Minimum number of source nodes* of the graphs (integer number)	$N_s/2$
$\mathbf{N_t}$	*Maximum number of target nodes* of the graphs (integer number)	
$\mathbf{n_t}$	*Minimum number of target nodes* of the graphs (integer number)	$N_t/2$
$\mathbf{N_a}$	*Maximum number of activity nodes* of the graphs (integer number).	
$\mathbf{n_a}$	*Maximum number of activity nodes* of the graphs (integer number).	$N_a/2$
Input degree of activity nodes, i.e. the number of edges incoming activity nodes (excepting wrappers, which have degree 1). In order to generate heterogeneous nodes, the input degree is randomly generated given the intervals for the random selection, namely:		
d	*Minimum degree* (integer number)	1
D	*Maximum degree* (integer number)	3
Property values for nodes and edges. In order to generate different values, they are randomly generated given the intervals for the random selection. Furthermore, as different properties may have different significant values, intervals are specified for each property, namely:		
$\mathbf{p_a}$	*Minimum actual freshness* of a source node (integer number)	0
$\mathbf{P_a}$	*Maximum actual freshness* of a source node (integer number)	24
$\mathbf{p_e}$	*Minimum expected freshness* of a target node (integer number)	24
$\mathbf{P_e}$	*Maximum expected freshness* of a target node (integer number)	48
$\mathbf{p_c}$	*Minimum processing cost* of an activity node (integer number)	0
$\mathbf{P_c}$	*Maximum processing cost* of an activity node (integer number)	2
$\mathbf{p_d}$	*Minimum inter-process delay* of an edge (integer number)	0
$\mathbf{P_d}$	*Maximum inter-process delay* of an edge (integer number)	4

Table 5.5 – Generation parameters

Data set	$N=N_s+N_t+N_a$	G	S	T	d	D
Small	from 10 to 50	from 10 to 100	10	10	1	3
Bulk	15	from 100 to 5000	10	10	1	2
Big	from 100 to 1000	from 100 to 1000	200	200	1	3

Table 5.6 – Generation parameters for the data sets

4.1.2. Generation correctness

The small data sets were analyzed in order to check whether well-formed graphs were generated and visualize them. Figure 5.12 shows some of the graphs generated in a test (with $10<N\leq20$) and Figure 5.13 shows a bigger graph generated in another test (with $40<N\leq50$). The generator takes no care of node positions but allows users to reposition nodes (drag and drop) in order better visualize them. Repositioning, even if possible, is not viable for huge graphs[*].

A first remark is that some graphs are not connected. We analyzed the possibility of checking connection during generation (and adding some edges to connect disconnected sub-graphs), but we discarded the idea because of the cost of such checking. Disconnection is not a problem for quality evaluation algorithms since data quality is propagated along existing data flow. So, we permitted the generation of disconnected sub-graphs. However, we

[*] Screen captures correspond to the graphical interface of DQE version 2, differing from that of the current version shown in Sub-section 2.3.

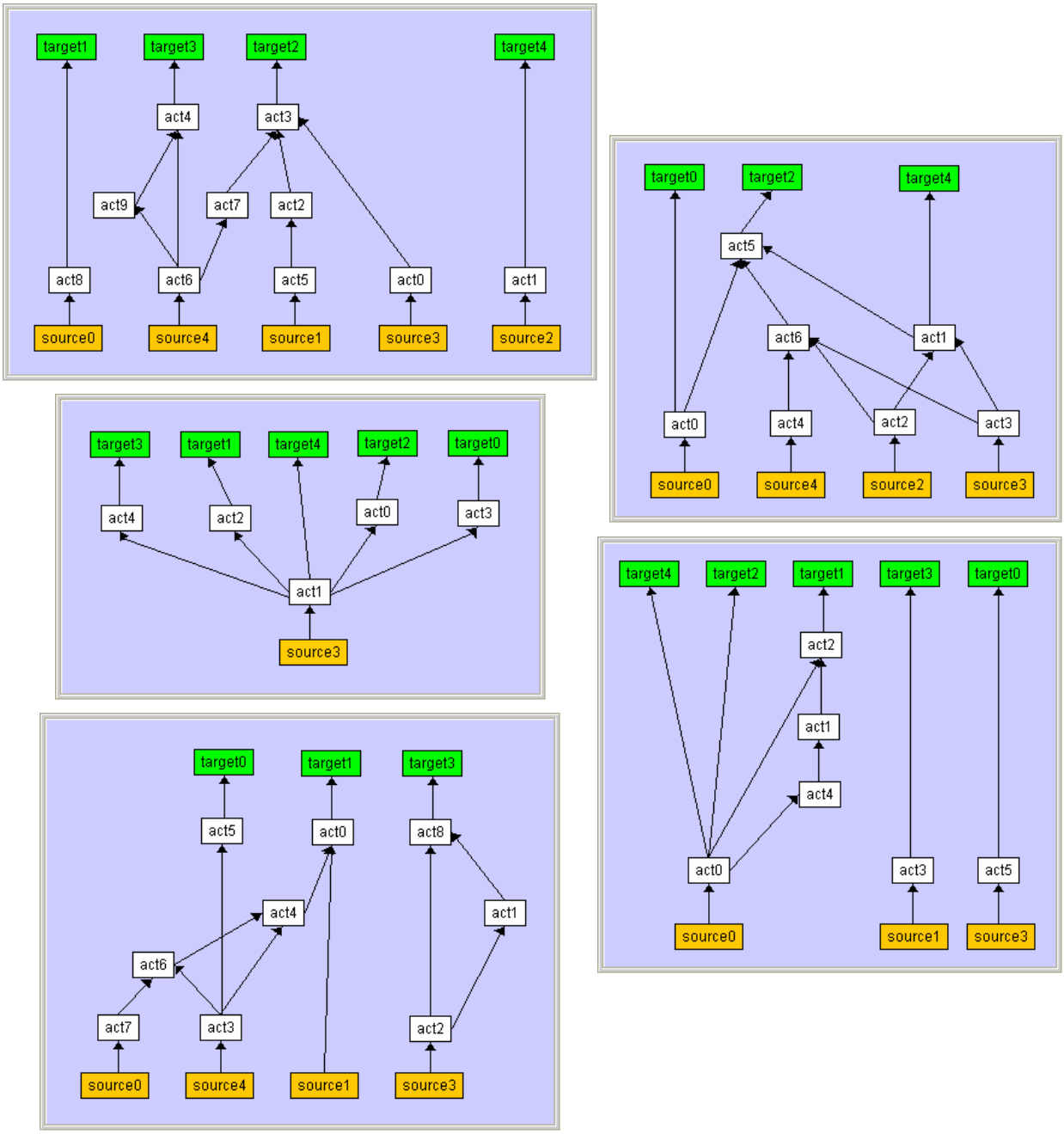

Figure 5.12 – Some generated graphs having among 10 and 20 nodes

assured that graphs respect the quality graph definition (Definition 3.4), i.e., graphs are acyclic, source nodes have no incoming edges and a unique outgoing edge, target nodes have a unique incoming edge and no outgoing edges. We also checked that all nodes and edges have the appropriated property values.

Property values were also verified, as well as the result of applying evaluation algorithms. The analysis of critical paths for the graph of Figure 5.13 is shown in Figure 5.14. Some big data sets were also analyzed (it was a tedious task). No figures are shown because graphs are too big to be displayed inside a screen.

A batch script, without graphical interface, was implemented to run the tests. It firstly invokes the generator indicating the name of a file containing generation parameters, it loads the generated session into the DQE tool, and it execute the *AcutalFreshnessPropagation* algorithm (presented in Chapter 3) for all generated graphs. Generation, loading and evaluation times (and other indicators) are logged in a file. All tests where repeated three times. In next sub-sections we discuss test results for application limits, quality evaluation performance and loading performance.

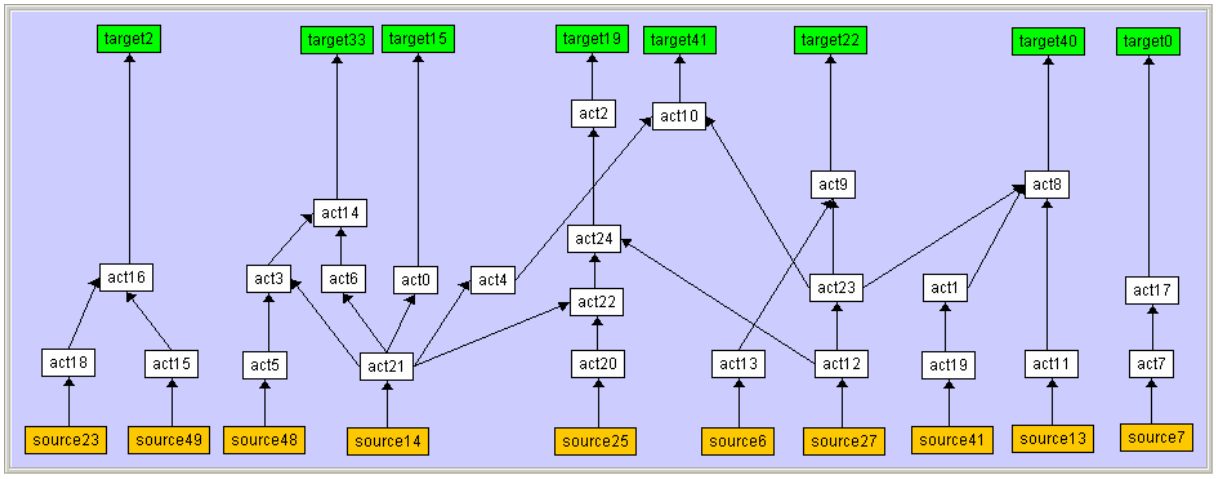

Figure 5.13 – Generated graph having 9 target nodes, 25 activity nodes and 10 source nodes

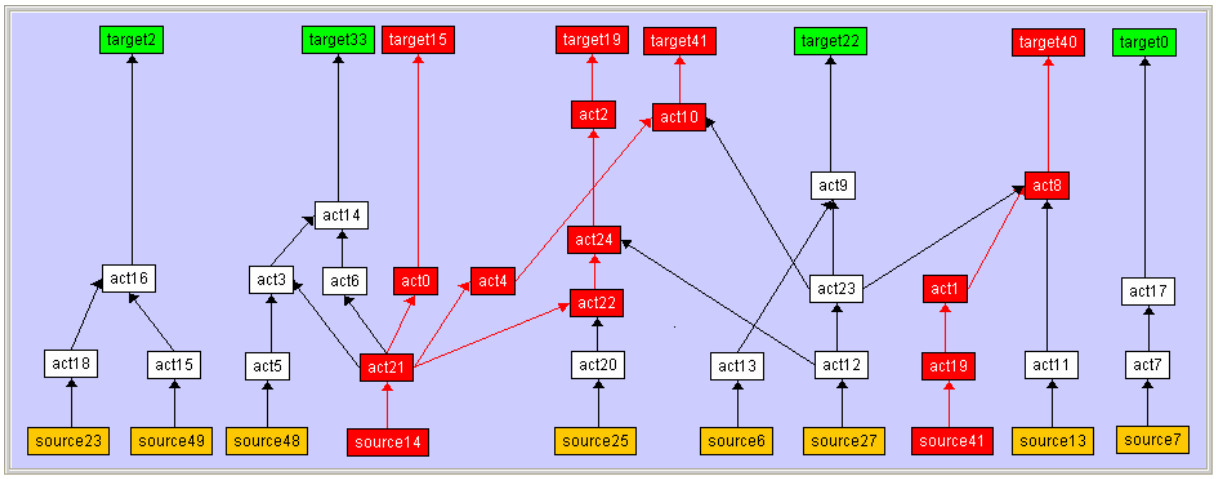

Figure 5.14 – Highlighting critical paths

4.2. Test of limitations

The bulk and big data sets where used to determine applications limits, i.e. know the number of graphs (with different topologies) supported concurrently by the tool. Remember that DQE version 2 loads data in memory, so, the test indicates how many graphs can be stored in memory for quality evaluation.

The result with the bulk data sets is that up to 2800 small graphs (15 nodes) can be treated in parallel; 2900 graphs can be loaded but memory overflow occurs during quality evaluation.

The result with the big data sets is shown in Table 5.7; colored cells indicates the data sets that were successful for all test repetitions, shadow cells indicate the data sets that were successful for all repetitions of the loading test but not for the evaluation test and uncolored cells indicates the data sets for which the tests failed for at least one repetition. Obviously, some data sets were not tested because of the fail of smaller data sets. The test shows that the tool supports data sets consisting in several hundreds graphs with several hundreds nodes. The tool can also support almost a thousand medium graphs (with a hundred nodes) or some huge graphs (with a thousand nodes). The application does not scale to thousands of huge graphs. Scalability must be provided replacing the memory storage by access to secondary storage, which is discussed as perspective in Chapter 6.

Next sub-section analyzes performance for the tests data sets that succeeded.

N \ G	100	200	300	400	500	600	700	800	900	1000
100										
200										
300										
400										
500										
600										
700										
800										
900										
1000										

Table 5.7 – Application limits for the big data sets

4.3. Test of performance

In this sub-section we analyze the tool performance for the three types of data sets. The main goal is to evaluate quality evaluation performance, but as quality graphs have to be loaded in memory in order to evaluate their quality, we also evaluate loading performance.

4.3.1. Quality evaluation performance

In this test we analyze quality evaluation performance. We measure the time it takes the tool to evaluate data freshness, i.e. executing the *ActualFreshnessPropagatiion* algorithm on all graphs of the data set.

The test with the small data sets shows that evaluation time is linear in the number of graphs and the number of nodes. This result is reasonable because the evaluation algorithm is applied to each graph, traversing each node once. Figure 5.15 shows the test result; the noise in the graphics is explained by the small magnitude of evaluation times (milliseconds), which makes difficult the filtering of other operation system routines that also consume time. The bigger data set (100 graphs of 50 nodes) was computed in 0.22 seconds.

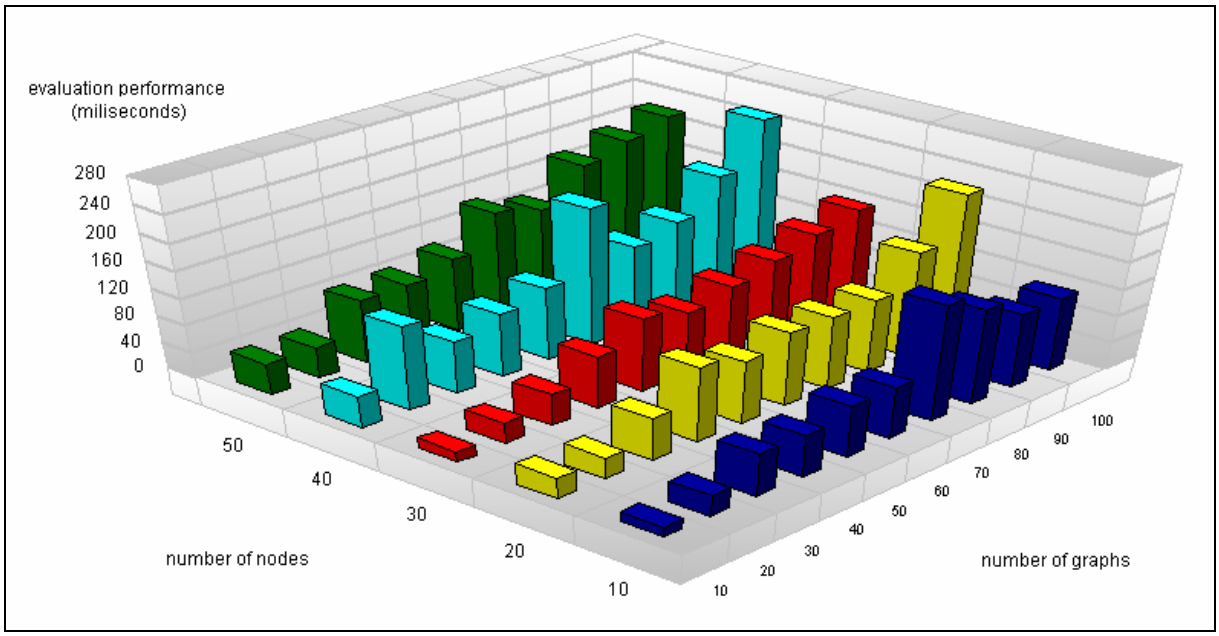

Figure 5.15 – Evaluation times for the small data sets

The test with the bulk data sets also shows that evaluation time is linear in the number of graphs. Figure 5.16 shows the test result. Quality evaluation can be performed in more than a thousand graphs in a second.

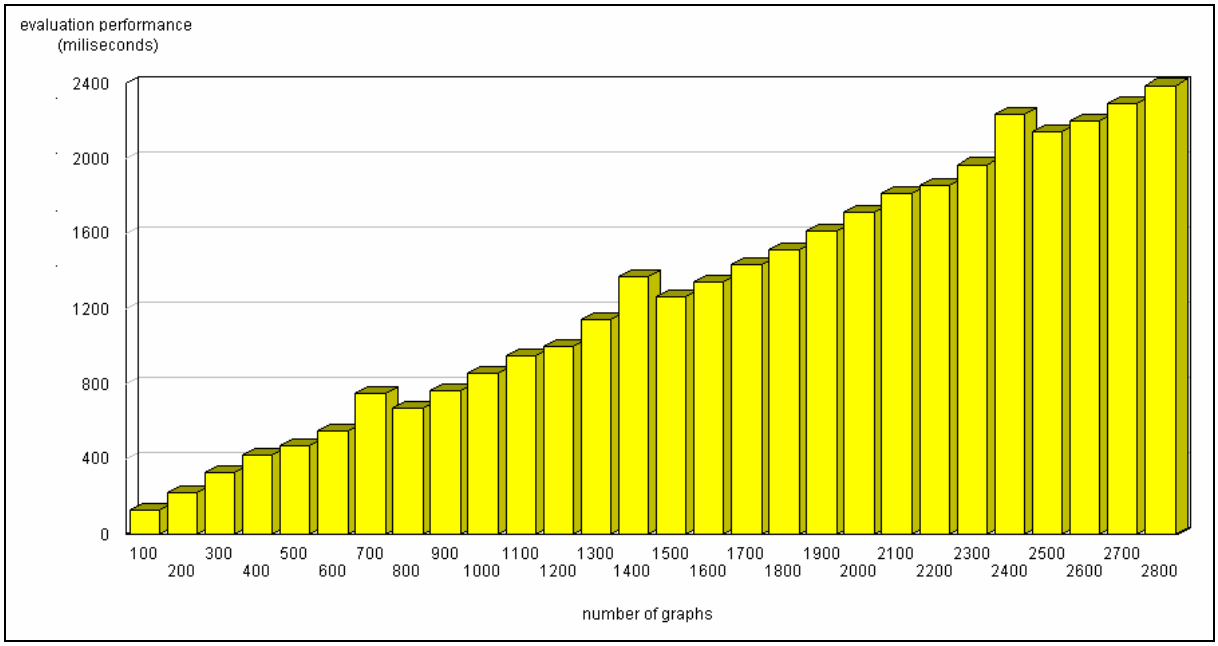

Figure 5.16 – Evaluation times for the bulk data sets

The test with the big data sets confirms that evaluation time is linear in the number of graphs and the number of nodes also for big graphs. However, the limit data sets (the biggest ones being supported) provoke a great number of memory faults, which cause a huge evaluation time, overdrawing the linear scale. Figure 5.17 show the test result. Quality evaluation in some hundred graph of some hundred nodes lasts some seconds.

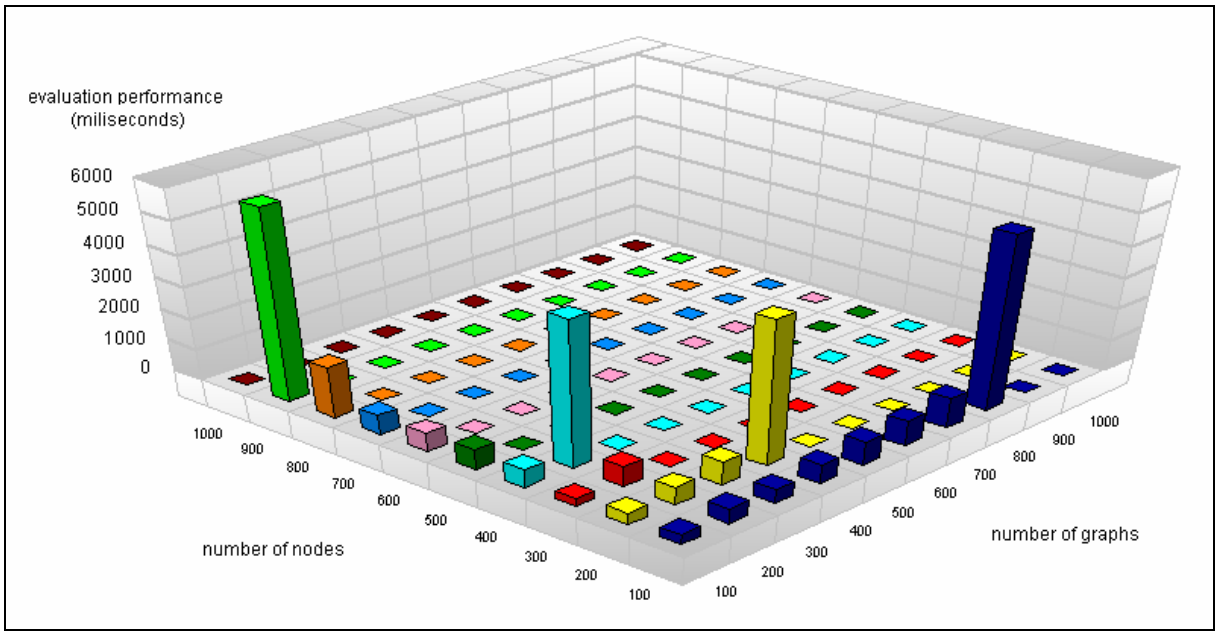

Figure 5.17 – Evaluation times for the big data sets (the bar corresponding to <900nodes,100graphs> overdraws the scale; its evaluation time is 40 seconds)

4.3.2. Loading performance

In this test we analyze loading performance. We measure the time it takes the tool to communicate with the metabase for loading each session and their components.

The test with the small data sets shows that loading time is linear in the number of graphs and slightly increases with the number of nodes. Figure 5.18 shows the test result. Times are bigger than evaluation ones; some data sets were loaded in almost a minute. Noise is also bigger, which is mainly due to Oracle internal routines (e.g. logging) and inter-process communication delays.

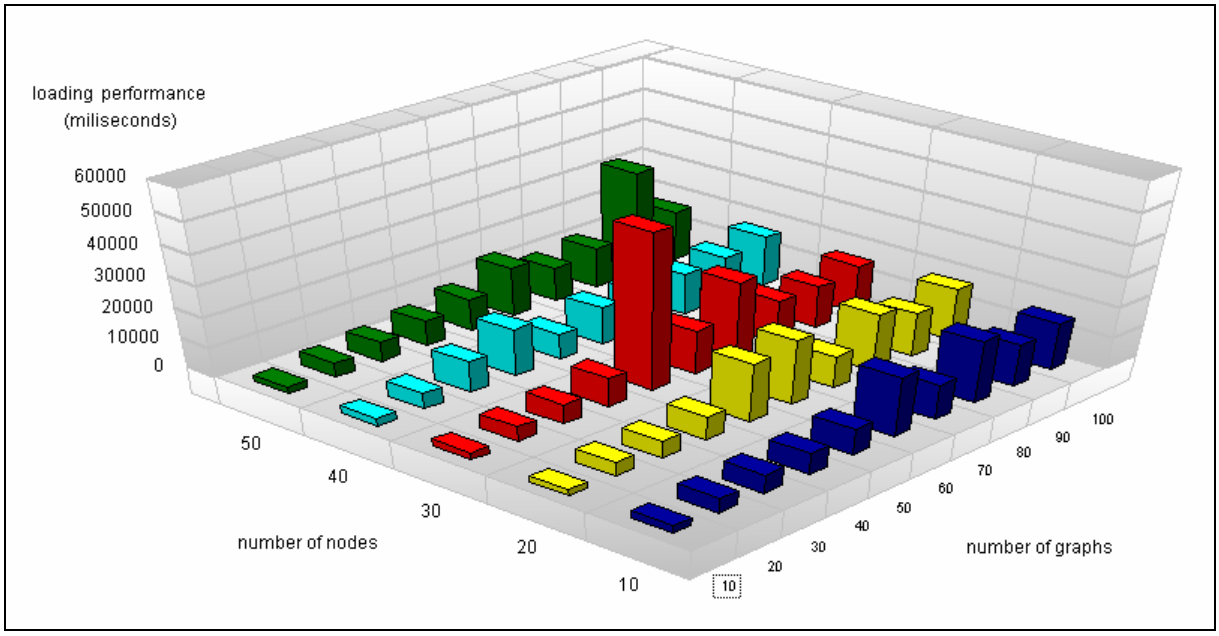

Figure 5.18 – Loading times for the small data sets

The test with the bulk data sets also shows that loading time is linear in the number of graphs. Figure 5.19 shows the test result. Loading can be performed in less than a minute for most data sets.

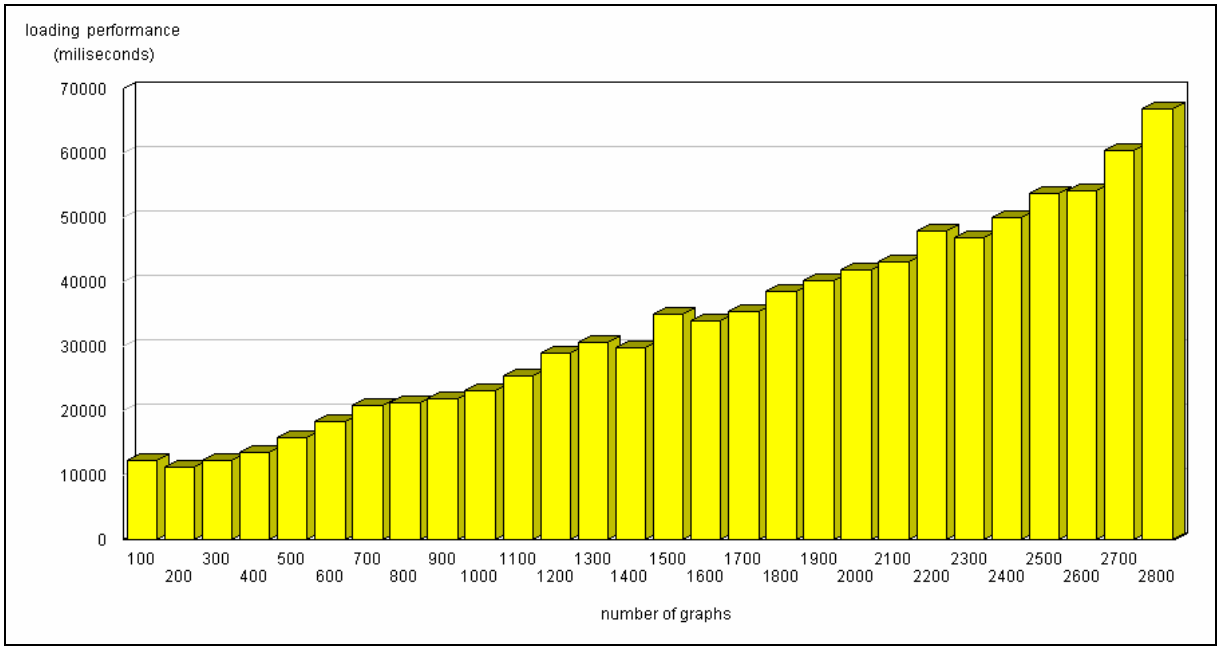

Figure 5.19 – Loading times for the bulk data sets

The test with the big data sets confirms that loading time is linear in the number of graphs and the number of nodes also for big graphs. However, the limit data sets (the biggest ones being supported) overdraw the linear scale. Figure 5.20 show the test result. The loading of some hundred graphs of some hundred nodes lasts some minutes.

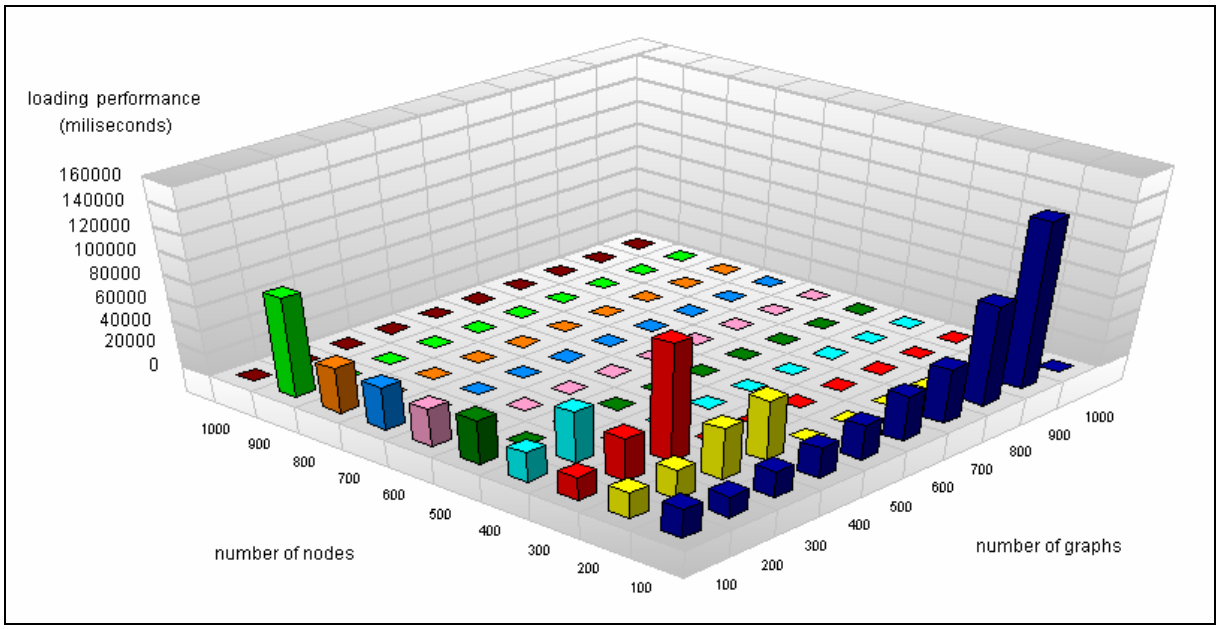

Figure 5.20 – Loading times for the big data sets

5. Conclusion

In this chapter we presented our experimentations with data freshness and data accuracy evaluation. We described the prototype of a data quality evaluation tool, DQE, which manages the proposed quality evaluation framework. The tools models the framework components, namely, data sources, data targets, quality graphs, properties, measures and quality evaluation algorithms.

We used the tool for evaluating data freshness and data accuracy in several application scenarios in order to validate our approach. Specifically, we described three applications: (i) an adaptive system for aiding in the generation of mediation queries, (ii) a web warehousing application retrieving movie information, and (iii) a data warehousing system managing information about students of a university. We showed how the DIS applications were modeled as quality graphs and how evaluation algorithms were instantiated to them. This experimentation allows validating the approach in real applications, specially, its ease of use for modeling DIS and properties and instantiating quality evaluation algorithms.

We also describe some tests for evaluating performance and limitations of the tool. We generated some data sets (quality graphs adorned with property values) and we executed a quality evaluation algorithm over each graph. The obtained results allow affirming that the tool can be used for large applications (modeling hundreds of graphs with hundreds of nodes each).

Chapter 6. Conclusions and Perspectives

This chapter presents our conclusions, highlighting contributions and discussing research perspectives. We also discuss the use of our proposal in some current research projects.

1. Summary and contributions

Data quality evaluation and assurance have been recognized as fundamental issues in Distributed Information Systems. Numerous works in the areas of Information System Design and Software Engineering deal with quality control and quality assurance. In a context of Data Integration Systems (DIS) providing access to large amounts of data extracted from alternative sources and conveying alternative query answers to users, information quality is becoming a *first class* property increasingly required by end-users. Some surveys and empirical studies have showed the importance of data quality for end users, in particular, when dealing with heterogeneous data coming from distributed autonomous sources. In addition, information quality problems have been reported as critical in several scientific and social areas such as Environment, Biology, Genetics, Commercial, Economy and Informatics in the Web.

In spite of the large number of research works dealing with data quality, several problems still remain to be solved. An analysis of the state of the art in data quality evaluation revealed that some technical issues have not been sufficiently treated. In particular, we highlight: the analysis of quality factors and metrics, the definition of user quality expectations, the assessment of source data quality and DIS property values, the evaluation of data quality, the enforcement of data quality, and the design of a DIS driven by data quality expectations.

This thesis deals with some of these topics and proposes a framework for data quality evaluation in data integration systems. We analyze two major quality factors: data freshness and data accuracy. We presented a taxonomy of freshness measurement techniques based upon the nature of data, the architectural techniques and the synchronization policies of the underlying DIS. We also proposed a taxonomy of accuracy measurement techniques based upon the granularity of measurement, the types of errors, the data types and the architectural techniques of the underlying DIS. Both taxonomies allow identifying the DIS properties that impact freshness and accuracy evaluation.

Such DIS properties are taken into account by quality evaluation algorithms in order to estimate the quality of the data conveyed to users in response to their queries. Evaluation algorithms take as input the DIS processes and a set of values qualifying source data, DIS properties and user expectations and combine these values generating as output a value that qualifies the data conveyed to the users. To this end, we model the different elements involved in data quality evaluation in a quality evaluation framework. Among these involved elements there are: data sources, data targets, DIS processes, DIS properties, quality measures and quality evaluation algorithms. In particular, we model DIS processes as direct acyclic graphs, called quality graphs, which have the same workflow structure than the DIS and are labeled with the DIS properties that are relevant for quality evaluation. Quality evaluation algorithms are based on the graph representation and consequently, the quality evaluation problem turns into value aggregation and propagation through this graph. We propose two types of algorithms: (i) for propagating quality actual values (from sources to targets) and (ii) for propagating quality expected values (from targets to sources). The former are used for estimating the quality of delivered data and the latter are used for determining quality constraints for source providers. The framework provides a flexible environment, which allows specializing evaluation algorithms in order to take into account the characteristics of specific application scenarios.

The proposed freshness evaluation algorithms take into account source data freshness, processing costs of DIS activities and inter-process delays among them. They can be instantiated for different application scenarios by analyzing the properties that influence source data freshness, processing costs and inter-process delays in those scenarios. We also propose different kinds of improvement actions to enforce data freshness when user expectations are not satisfied. Such actions are building blocks that can be composed to improve data freshness in concrete DISs. Our enforcement approach supports the analysis of a DIS at different abstraction levels in

order to identify critical points of the DIS and to target the study of improvement actions for these critical points. The graph representation of the DIS allows the rapid visualization of such critical points.

The proposed accuracy evaluation algorithm takes into account the distribution of inaccuracies in source relations. Specifically, source relations are partitioned in areas and sub-areas having homogeneous accuracy. User queries are rewritten in terms of areas and accuracy values associated to those areas are propagated through query rewritings. As a result, we obtain a partition of query results according to data accuracy. An accuracy value is also aggregated for the whole query result. We also propose some improvement actions for enforcing data accuracy. The proposed actions consist in filtering areas and sub-areas not satisfying accuracy expectations and in incrementally conveying data according to their accuracy (e.g. displaying first the most accurate areas). This represents an improvement to source selection proposals because accurate areas of several source relations can be combined while discarding inaccurate areas (instead of discarding whole source relations).

Finally, we developed a prototype of a quality evaluation tool, called DQE, which implements the proposed quality evaluation framework. The tool allows displaying and editing the framework components as well as executing quality evaluation algorithms. The prototype was used for evaluating data freshness and data accuracy in three applications: (i) an Adaptive Mediation application delivering data about scientific publications, (ii) a Web Warehousing application retrieving movie information, and (iii) a Data Warehousing application managing information about students of a university. This experimentation allowed the validation of the approach in real applications and raised the attention on the practical difficulties of modeling DISs as quality graphs, setting property values and instantiating evaluation algorithms. In addition, we described some tests for evaluating performance and limitations of the tool which proved that it can be used for large applications, modeling hundreds of graphs with hundreds of nodes each.

As summary, the main contributions of this thesis are:

- ***A detailed analysis of data freshness and data accuracy quality factors.*** The main result of this analysis is a survey on data freshness and data accuracy definitions and metrics used in the literature, which intends to clarify the meanings of such quality properties. We also elaborated a taxonomy of elements that influence their evaluation. Additionally, this analysis highlights major research problems which still remain to be solved. As far as we know, such deep analysis has not yet been done for these quality factors.
- ***The proposal of techniques and algorithms for the evaluation and enforcement of data freshness and data accuracy.*** Our contribution consists in the specification of evaluation algorithms and improvement policies for data freshness and data accuracy. The definition of a homogeneous framework to manipulate quality factors constitutes a basis to the identification of the DIS properties that impact freshness and accuracy evaluation. We proposed some evaluation algorithms that consider such properties and some improvement actions that intend to achieve quality expectations.
- ***A prototype of tool intended to be used in practical contexts of DIS management.*** The main results concerning the implementation of the proposed framework are the specification and prototyping of a quality evaluation tool that manages the framework. The framework components are specified in an abstract model, which supports the dynamic incorporation of new components to the tool, especially the inclusion of new quality factors and their evaluation algorithms. This brings support for the extensibility of the tool regarding the evaluation of other quality factors. The operational-style of the proposal, in particular the graph representation of DIS processes and the specification of quality evaluation algorithms as graph propagation methods facilitate their reuse for a wide range of DIS applications.

In next section we discuss some research perspectives.

2. Perspectives

In this section we discuss some improvements that may be done to our proposal in order to complete the analysis of data freshness and data accuracy and provide extensibility to the quality evaluation framework. We aim to treat these issues as future work. In addition, the research topics analyzed in this thesis suggest further research perspectives. Next sub-sections discuss both, near future work and research perspectives and discuss the use of our proposal in some current research projects.

2.1. Near future work

In near future, we aim to improve some features of the quality evaluation tool and perform additional performance tests. In addition, we aim to analyze the relationship among data freshness and data accuracy quality factors. We describe these perspectives:

Improvement of the DQE tool

In Chapter 5, we described a prototype of the DQE tool, which implements the quality evaluation framework and allows the execution of quality evaluation algorithms over quality graphs. The functionalities of the tool were described in Sub-section 2.1 of Chapter 5. However, further functionalities are only partially provided or are scheduled for future versions. Concretely, we want to extend DQE with the following functionalities:

- Navigation in a hierarchy of quality graphs: Currently, the tool allows visualizing several quality graphs, which may represent the DIS at different abstraction levels. However, the methods for navigating in the hierarchy of graphs (level-up and level-down, as well as zoom+, zoom–, focus+ and focus–) are not yet implemented. Hence, the tool brings limited support to the top-down analysis approach presented in Chapter 3.
- Graphical representation of partitions: Partitions are treated as property labels (as explained in Sub-section 4.4 of Chapter 4) so they can be edited and displayed as formatted text. However, we should provide a graphical interface for visualizing partitions, e.g. coloring sub-areas. In addition, areas not satisfying accuracy expectations can be graphically highlighted in order to quickly visualize critical points of quality graphs (this is analogous to the coloring of critical paths for data freshness).
- Implementation of a library of improvement actions: By the moment, only elementary improvement actions (described in Sub-section 4.4 of Chapter 3) can be applied. We aim to provide a collection of macro actions as well as the mechanisms for developing new actions by composing elementary ones.

Concerning scalability, we aim to modify the persistency mechanism of the tool in order to support quality evaluation in larger applications. Test results presented in Chapter 5 showed that the tool can be used for large applications (modeling hundreds of graphs with hundreds of nodes each). However, the application does not scale to thousands of huge graphs. Scalability must be provided replacing the memory storage by the access to secondary storage. Remember that the second version of DQE (used in the tests) loads all quality graphs in memory. In the third version, we experimented a new loading mechanism, in which quality graphs are loaded in memory only when they are used (either for editing the graph or for executing an evaluation algorithm over it) and memory is liberated thereafter. As expected, this new mechanism is less performing, which is unnecessary paid in small applications. For that reason, we aim to improve the implementation of the persistency layer in order to cache, in main memory, a certain number of quality graphs (possibly all, in the case of small applications). Such cache should be reactive and proactive, i.e. it should contain the quality graphs that have been recently used as well as load the quality graphs that will be used soon (e.g. when executing an algorithm in batch mode we know which are the following graphs that will be accessed).

Test of performance and precision of the accuracy evaluation algorithm

In Chapter 5 we presented the results of some tests performed over DQE in order to determine the number of quality graphs that can be loaded simultaneously and the performance of executing the freshness evaluation algorithm over such graphs. We plan to perform similar tests for the accuracy evaluation algorithm.

Additionally, in Chapter 4 we observed that the precision of obtained accuracy values relies on the techniques used for measuring accuracy of source relations, partitioning them and estimating the selectivity of rewritings. We want to investigate which is the influence of such techniques, for example, which is the gain in precision if we use sophisticated statistical models instead of simple histograms for selectivity estimation. Our proposal for accuracy evaluation can be compared to the proposal of [Naumann+1999] in order to measure in which degree the partitioning of source relations allows obtaining a better approximation of the accuracy of query results. We should also compare the approach with the proposal of [Rakov 1998] which measures accuracy of the exact query result.

Exploring the relation between data freshness and data accuracy quality factors

In this thesis we studied, separately, freshness and accuracy quality factors, identifying the DIS properties that impact their evaluation. However, these quality factors are not independent, and hence, actions for improving one factor may have side-effects (improve or degrade) on the other. For example, some actions for improving data freshness (e.g. reducing processing costs of activities or refreshing materialized data more frequently) may also improve semantic correctness because they reduce the amount of obsolete data (data that no longer represent real-world because of changes in real-world). Conversely, corrective actions for enforcing data accuracy (e.g. data cleaning or format standardization) may consume significant time and degrade data freshness. In addition, in some application contexts cleaning processes cannot be executed on-line forcing data materialization, which also degrades data freshness.

A trade-off between accuracy and freshness is necessary. Such trade-off should take into account the relationship among freshness and accuracy factors. Table 6.1 sketches some correlations among them. However, as specific improvement actions can be defined for specific scenarios, specific relations may be analyzed for such scenarios. We made a preliminary analysis of correlations in some particular scenarios. We aim to formalize such analysis in near future and obtain quality improvement guidelines that take into account both factors.

Freshness factors / **Accuracy factors**	**Currency / Timeliness**
Semantic correctness	- Improving data freshness reduces expired data → improves data semantic correctness - Correcting errors takes time → degrades data freshness
Syntactic correctness	- Constraints (as format or belongness to a referential) are not time-related, so improving data freshness does not affect data syntactic correctness - Correcting errors takes time → degrades data freshness
Precision	- Improving data freshness does not affect data precision - Improving activities to avoid losing precision may take time → degrades data freshness

Table 6.1 – Correlations among freshness and accuracy factors

2.2. Other research perspectives

This thesis proposed some techniques and algorithms for quality evaluation and quality enforcement that we aim to extend in three lines: (i) the analysis of further quality factors and their inter-relationships, (ii) the development of specialized quality enforcement strategies, (iii) the application of the proposed framework to further application scenarios. We describe several research perspectives in these directions:

Generalization of the framework

In Chapter 3 we presented a quality evaluation framework, which allows modeling the DIS properties involved in data freshness evaluation and developing freshness evaluation algorithms. In Chapter 4 we extended the framework for evaluating data accuracy, representing further DIS properties and developing a new evaluation algorithm.

For accuracy evaluation we focused in a mediation scenario (relational model, JSP queries) and thus, we proposed evaluation and enforcement techniques for this scenario. As future work, we hope to extend the approach to consider other scenarios, especially those having activities for correcting errors (e.g. data cleaning and format standardization routines). The Data Warehouse application described in Chapter 5 can be taken as a case of study for proposing evaluation techniques and algorithms for this type of scenarios. The evaluation approach proposed in Chapter 4 is a first step towards the definition of general evaluation methods that might be instantiated to several types of application scenarios, as we proposed for data freshness.

On the other hand, the idea of partitioning relations can be reused for data freshness, i.e. we can partition source relations (and query results) in areas having homogeneous freshness. In this line, the analysis of the relationship

between freshness and accuracy involves an additional challenge: the partitioning of relations according to several quality factors.

Additionally, we think that the framework can be reused for evaluating other quality factors. To this end, we should analyze the DIS properties that impact those factors and we should implement the appropriate evaluation algorithms. Preliminary experiments performed with the Adaptive Mediation application (presented in Chapter 5) showed that such extension is feasible for certain quality factors, but of course, detailed surveys, as those presented in this thesis for data freshness and data accuracy, should be done for further quality factors. The abstract model of DQE, which supports the dynamic incorporation of new components to the tool (especially the inclusion of new quality factors and their evaluation algorithms) brings support for the extensibility of the tool regarding the evaluation of other quality factors.

Data quality in a context of information personalization

Personalization and quality of information are two major challenges for computer science industry. Relevance, intelligibility and adaptability of retrieved information are the key factors for success or rejection of many information retrieval systems. In this context, adaptability means that usual practices and preferences of users must be taken into account. Kostadinov's thesis analyzes the most important knowledge that should compose a user profile and proposes a generic profile model, which includes quality preferences [Kostadinov 2006]. The analysis of the quality factors that are the most relevant for end users and the definition of evaluation techniques adapted to their environments become important challenges for data personalization.

Another important issue is the comparison of actual quality values (aggregated from source quality values) with user quality expectations (expressed in user profiles). This comparison can be done at different moments: (i) during query rewriting, in order to reformulate queries taking into account user profiles, for example, to include further predicates for representing user preferences; (ii) during query execution, in order to consider the sources satisfying quality expectations; and (iii) during delivery of query results, in order to order results according to their quality. Some ideas presented in Chapter 4, specifically those concerning improvement actions, can be applied to this context. Analogous improvement actions should be defined for other quality factors.

Other aspects of user preferences (e.g. the relative importance of quality factors for users) should be modeled in user profiles and used in the personalization process. The filtering of data according to multi-criteria conditions should be also analyzed.

Integration with quality assessment techniques

Even the DIS and its relevant properties can be easily modeled in DQE, the assessment of source data quality and DIS property values may be a tedious task and may imply the development of specialized routines. In this thesis we do not dealt with the assessment of source data quality nor DIS property values, but we found that the analysis of assessment techniques can be an interesting line for future works.

Many enterprises are interested in the development of web-services or plug-ins in order to obtain statistics of DIS properties and invoke quality evaluation algorithms. In some cases (e.g. a telecommunications company having 900 servers) the automation of assessment techniques is essential. To achieve this, it is necessary to develop an extensible model for representing the relevant statistics, the logs that should store them and the processes for aggregating property values from logs. An example on the use of statistics and logs were presented in Sub-section 3.2 of Chapter 5 for Web Warehousing applications.

Development of specialized quality enforcement techniques

In this thesis we suggested basic improvement actions that are general enough to be applied to different types of DIS but their use for quality enforcement should be guided by some high-level strategies in order to be effective. In particular, the selection of the most appropriate actions for a given DIS may depend on addition criteria as DIS configuration, reengineering costs and user preferences. A wide range of improvement strategies (based on improvement actions) can be defined for specific DISs. Among these strategies, we are particularly interested in many ones that we partially explored during this thesis:

- Synchronization of activities: In Section 5 of Chapter 3 we analyzed the synchronization of activities in a concrete application scenario in order to reduce inter-process delays among activities and therefore improve data freshness. Similar analysis can be carried out for other application scenarios with different characteristics.
- Quality-driven data reconciliation. In the context of Web Warehousing applications (described in Sub-section 3.3 of Chapter 5), we experimented the incorporation of freshness values to disambiguate conflicts among data. The proposal of reconciliation policies that consider further quality factors becomes an interesting challenge.
- Quality-driven ordering and filtering of query answers: In Chapter 4 we proposed partitioning query results and filtering areas with low accuracy. Note that the most restrictive are users' accuracy expectations, the smallest is the result. Accuracy expectations should be balanced with completeness expectations for avoid filtering too much data and conveying a representative set of tuples to users. An alternative consists in ordering areas according to accuracy and incrementally deliver areas (the most accurate first), which allows users to dynamically decide the amount of data they want to analyze. The trade-off among data freshness, data accuracy and other quality factors should be also analyzed.
- Selective rewriting of user queries. Data filtering can be performed in early stages of query planning, precisely, during the rewriting of user queries in terms of partitions. In this context, sub-areas providing data with low accuracy can be excluded for query buckets and thus they will not be used in query rewritings. This strategy allows extracting the most accurate data of each source relation instead of discarding whole source relations. Analogously, the trade-off among accuracy, completeness and other quality factors should be analyzed.
- Quality-driven generation of mediation queries: In the Adaptive Mediation application described in Chapter 5, we described the use of data quality values for selecting the mediation query that best adapts to user quality expectations. Such selection is carried out by generating a set of candidate queries, evaluating their quality and comparing it with user profiles. Additionally, data quality can be used to improve query generation by eliminating intermediate results that cause the non-satisfaction of quality expectations. This filtering may considerable reduce the search space of the generation algorithms (proposed in Xue's thesis [Xue 2006]) and thus optimize generation performance. Furthermore, the generation of a limited number of queries also reduces the time spent in selecting the most appropriate one.

Although we have shown that our approach can be used for DIS design, maintenance and evolution, we don't treat these topics in this thesis. The thesis of Marotta [Marotta 2006], based in our framework, treats the problem of detecting changes in source data quality and propagating changes to the DIS. Its main goal is to propose techniques for maintaining as much as possible the satisfaction of users' quality requirements. On the one hand, she proposes a proactive strategy based on probabilistic techniques, which allow to model source quality behavior and to calculate the quality reliability of the system. On the other hand, she proposes a reactive strategy that must be applied for compensating source quality changes.

2.3. Towards quality-driven design of DIS

Figure 6.1 positions our proposal among some of the technical issues discussed as perspective. The proposed quality evaluation algorithms take as input the DIS processes and a set of values qualifying source data, DIS properties and user expectations. The acquisition of such values should be carried out by quality assessment, property assessment and profile management modules. Such modules should take into account the analysis of DIS and source properties that impact in quality evaluation. Thus, evaluation algorithms read property values from a metadata repository and combine them, obtaining quality values for the data conveyed to users. DIS processes adorned with quality values are the input for quality enforcement techniques, which propose improvement actions for DIS design. Quality-driven design methodologies should apply these improvement actions in the design or maintenance of DIS processes.

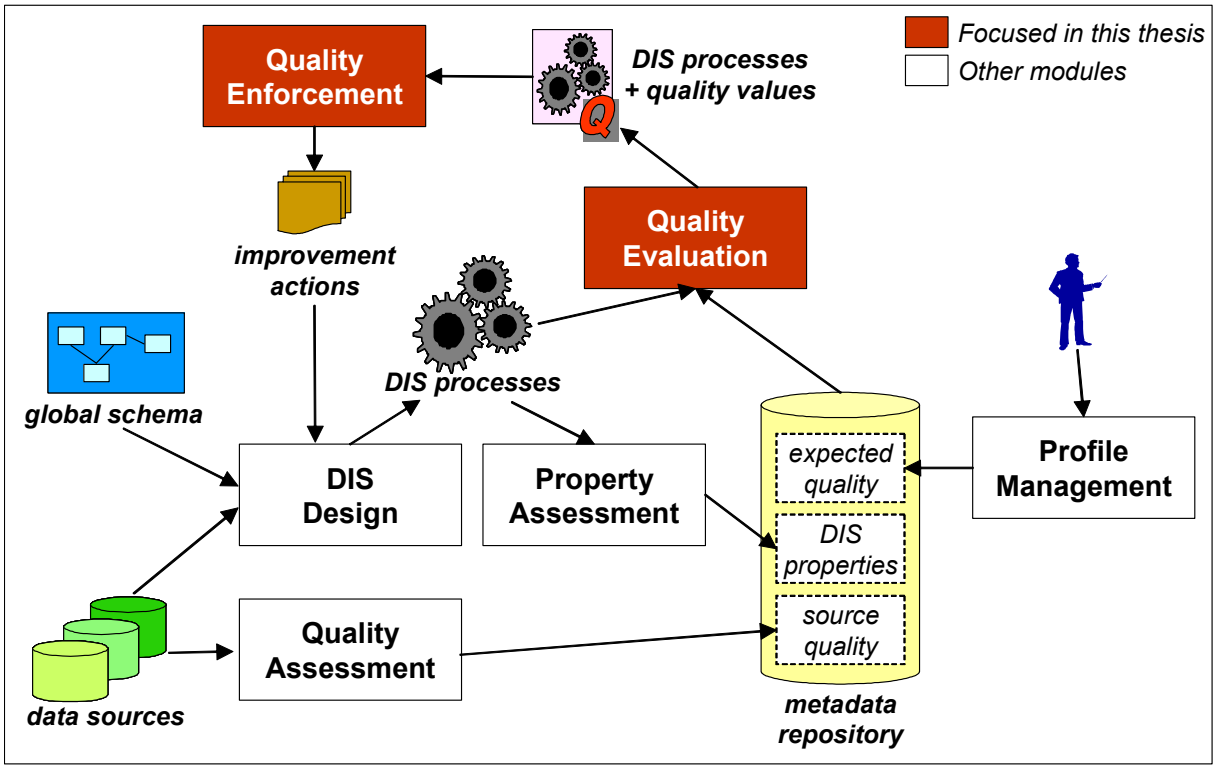

Figure 6.1 – Summary of the proposed solutions and perspectives

We are currently addressing these problems in some research projects: In the APMD* project, we will apply quality evaluation techniques and improvement actions to the context of data personalization. The main goal of the project is to carry out a cross investigation about personalization and quality of information. More precisely, its objectives are to propose formal models and robust algorithms, capable of capturing users' preferences which will be represented in user profiles and will be used for information filtering and adaptive display in a large scale environment. Concerning data quality, the project makes emphasis in the joint impact of profiles and information quality on the relevance of query results. A related ongoing thesis [Kostadinov 2006] proposes a taxonomy of the most important knowledge composing a user profile (including quality properties) and defines a generic profile model that can be instantiated and adapted for each specific application. In addition, a CSIC project† addresses quality evaluation and enforcement from a theoretical point of view, dealing with the analysis of several quality factors (including data freshness and data accuracy) and the proposition of techniques for quality-driven design and change management. An ongoing thesis [Marotta 2006] extends the quality evaluation framework with change management techniques. In the Quadris project‡ we analyze several quality factors and study their evaluation in several applications in the biomedical, commercial and geographical domains. The objective of the project is to solve the various data quality problems that appear when modeling DISs, when integrating and querying multi-source data and when evaluating data quality.

Finally, a cooperation agreement was signed with the Pasteur Institute§ in order to build a Laboratory Information Management System (LIMS). One of the technical aspects to be addressed is information quality management. There are two master thesis ongoing in this issues, one of them extending the quality evaluation framework with quality factors qualifying web services (e.g. avalaibility and performance), and the other one focusing in the construction of a decisional system for manipulating genetic data (including quality properties).

---

* Research project: "APMD - Accès Personnalisé à des Masses de Données" (Personalized access to massive data), ACI MASSES DE DONNEES - PROJET MD33/04-07, French Ministry of Research, France, 2004-2007. URL: http://apmd.prism.uvsq.fr/

† Research project: "Análisis de factores de calidad en sistemas de información multi-fuentes" (Analysis of quality factors in multi-source información systems), Instituto de Computación – Universidad de la República, financed by Comisión Sectorial de Investigación Científica (CSIC), Uruguay, 2005-2007. URL: http://www.fing.edu.uy/inco/grupos/csi/esp/Proyectos/Calidad/index.html

‡ Research project: "Quadris – Qualité des données et des systèmes d'information multi-sources" (Quality of data and multi-source information systems), ACI MASSES DE DONNESS, French Ministry of Research, France, 2006-2009. URL: http://www.irisa.fr/quadris/

§ Cooperation agreement between Institute Pasteur Montevideo and Instituto de Computación – Universidad de la República, Uruguay, 2006-2008.

References

[Abiteboul+1998] Abiteboul, S.; Duschka, O.: "*Complexity of answering queries using materialized views*". In Proc. of the 1998 ACM Int. Symposium on Principles of Database Systems (PODS'98), USA, 1998.

[Acharya+1999] Acharia, S.; Gibbons, P.B.; Poosala, V., Ramaswamy, S.: "*Join synopses for approximate query answering*". In Proc. of the 1999 ACM Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, USA, 1999.

[Altareva 2004] Altareva, E.: "*Improving Integration Quality for Heterogeneous Data Sources*". PhD Thesis, Universität Düsseldorf , 2004.

[Amat+2005] Amat, G. ; Laboisse, B.: "*B.D.Q.S. Une gestion opérationnelle de la qualité de données*". 1st workshop on Data and Knowledge Quality (DKQ'2005), Paris, France, 2005.

[Ballou+1985] Ballou, D.; Pazer, H.: "*Modeling data and process quality in multi-input, multi-output information systems*". Management Science, Vol. 31 (2): 150–162, Feb. 1985.

[Ballou+1998] Ballou, D.; Wang, R.; Pazer, H.; Tayi, G.: "*Modelling Information Manufacturing Systems to Determine Information Product Quality*". Management Science, Vol. 44 (4), April 1998.

[Ballou+2003] Ballou, D.; Pazer, H.: "*Modeling Completeness versus Consistency Tradeoffs in Information Decision Contexts*". IEEE Transactions on Knowledge Data Engineering (KDE'2003), Vol. 15(1): 240-243, 2003.

[Baralis+1997] Baralis, E.; Paraboschi, S. Teniente, E.: "*Materialized view selection in a multidimensional database*". In Proc. of the 23rd Int. Conf. on Very Large Databases (VLDB'97), Athens, Greece, 1997.

[Berti-Equille 2004] Berti-Equille, L.: "*Un état de l'art sur la qualité des données*". Ingénierie des systèmes d'information (ISI), Hermès, Vol. 9(5-6) :117-143, 2004

[Bobrowski+1998] Bobrowski, M.; Marré, M.; Yankelevich, D.: "*A Software Engineering View of Data Quality*". 2nd Int. Software Quality Week Europe (QWE'98), Brussels, Belgium, 1998.

[Bouzeghoub+1999] Bouzeghoub, M.; Fabret, F.; Matulovic-Broqué, M.: "*Modeling Data Warehouse Refreshment Process as a Workflow Application*". In Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, 1999.

[Bouzeghoub+2002] Bouzeghoub, M.; Kedad, Z.: "*Quality in Data Warehousing*". Information and database quality, Piattini, M.; Calero, C.; Genero, M. (eds), Kluwer Academic Publisher, 2002.

[Bouzeghoub+2004] Bouzeghoub, M.; Peralta, V.: "*A Framework for Analysis of Data Freshness*" In Proc. of the 1st Int. Workshop on Information Quality in Information Systems (IQIS'2004), Paris, France, 2004.

[Braumandl 2003] Braumandl, R.: "*Quality of Service and Optimization in Data Integration Systems*". In Proc. Of GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW'2003), Leipzig, Germany, 2003.

[Breiman+1984] Breiman, I.; Friedman, J.; Olshen, R.; Stone, C.: "*Clasification and Regression Trees*". Wadsworth International Group, 1984.

[Bright+2002] Bright, L.; Raschid, L.: "*Using Latency-Recency Profiles for Data Delivery on the Web*". In Proc. of the 28th Int. Conf. on Very Large Databases (VLDB'02), Hong Kong, China, 2002.

[Bruno+2002] Bruno, N.; Chaudhuri, S.: "*Exploiting Statistics on Query Expressions for Optimization*". In Proc. of the 2002 ACM Int. Conf. on Management of Data (SIGMOD'2002) , Madison, USA, 2002.

[Calvanese+1999] Calvanese, D.; De Giacomo, G.; Lenzerini, M.; Nardi, D.; Rosati, R.: "*A principled Approach to Data Integration and Reconciliation to Data Warehousing*". In Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, 1999.

[Calvanese+2001] Calvanese, D.; Lembo, D; Lenzerini, M.: "*Survey on methods for query rewriting and query answering using views*". Technical Report, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 2001.

[Cappiello+2002] Cappiello, C; Francalanci, C; Pernici, B.: "*A Model of Data Currency in Multi-Channel Financial Architectures*". In Proc. of the 7th Int. Conf. on Information Quality (IQ 2002), Cambridge, USA, 2002.

[Chandra+1977] Chandra, A.; Merlin, P.: "*Optimal implementation of conjunctive queries in relational databases*". In Proc. of the 9th ACM Symp. On Theory of Computing (STOC'77), 1977.

[Chekuri+1997] Chekuri, C.; Rajaraman, A.: "*Conjunctive query containment revisited*". In Proc. of the 6th Int. Conf. on Database Theory (ICDT'97), 1997.

[Chirkova+2001] Chirkova, R.; Halevy, A.; Suciu, D.: "*A formal perspective on the view selection problem*". In Proc. of 27th Int. Conf. on Very Large Databases (VLDB'01), Roma, Italy. 2001.

[Cho+2000] Cho, J.; Garcia-Molina, H.: "*Synchronizing a database to improve freshness*". In Proc. of the 2000 ACM Int. Conf. on Management of Data (SIGMOD'00), pages 117-128, Dallas, USA. 2000.

[Cho+2003] Cho, J.; Garcia-Molina, H.: "*Estimating Frequency of Change*". ACM Transactions on Internet Technology, Vol. 3 (3): 256–290, August 2003.

[Cooper 1981] Cooper, M.: "*Survey of methods of pure nonlinear integer programming*". Management Science, Vol. 27: 353-361, 1981.

[Etcheverry+2005] Etcheverry, L.; Gatto, P.; Tercia, S.; Marotta, A.; Peralta, V.: "Análisis del proceso de carga del Sistema de Data Warehousing de Enseñanza de la Facultad de Ingeniería". Technical Report, InCo, Universidad de la República, Uruguay, 2005.

[Etcheverry+2006] Etcheverry, L.; Tercia, S.; Marotta, A.; Peralta, V.: "Medición de la Exactitud de Datos Fuente: Un caso de Estudio". Technical Report, InCo, Universidad de la República, Uruguay, 2006.

[Fajardo+2004] Fajardo, F.; Crispino I.: "*Implementación de una Plataforma para Análisis de Calidad de Datos*". Pre-grade Report, Facultad de Ingeniería, Universidad de la República, Uruguay, 2004.

[Fajardo+2004a] Fajardo, F.; Crispino, I.; Peralta, V.: "*DQE: Una Herramienta para Evaluar la Calidad de los Datos en un Sistema de Integración*". X Congreso Argentino de Ciencias de la Computación (CACIC'2004), La Matanza, Argentine, 2004.

[Fugini+2002] Fugini, M.; Mecella, M.; Plebani, P.; Pernici, B.; Scannapieco, M.: "*Data quality in cooperative web information systems*". Technical Report, 2002.

[Gal 1999] Gal, A.: "*Obsolescent materialized views in query processing of enterprise information systems*". In Proc. of the 1999 ACM Int. Conf. on Information and Knowledge Management (CIKM'99), pages 367-374, Kansas City, USA. 1999.

[Galhardas+2000] Galhardas, H.; Florescu, D.; Shasha, D.; Simon, E.: "*An extensible framework for data cleaning*". In Proc. of the Int. Conf. on Data Engineering (ICDE'2000), San Diego, Dallas, USA, 2000.

[Galindo+2004] Galindo, J.; Urrutia, A.; Piattini, M.: "*An Approach for Implementing Fuzzy Relational Databases in Classic DBMS*". In Proc. of the XII Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF'2004), Jaén, Spain, 2004.

[Gancarski+2003] Gancarski, S.; Le Pape, C.; Valduriez, P.: "*Relaxing Freshness to Improve Load Balancing in a Cluster of Autonomous Replicated Databases*". In Proc. of the 5th workshop on Distributed Data and Structures (WDAS), Thessaloniki, Greece. 2003.

[Gertz+1998] Gertz, M.; Tamer Ozsu, M.; Saake, G.; Sattler, K.: "*Managing Data Quality and Integrity in Federated Databases*". In Proc. of the 2nd Working Conf. on Integrity and Internal Control in Information Systems (IICIS'98), Warrenton, USA, 1998.

[Gertz+2004] Gertz, M.; Tamer Ozsu, M.; Saake, G.; Sattler, K.: "*Report on the Dagstuhl Seminar: Data Quality on the Web*". SIGMOD Record Vol. 33(1), March 2004.

[Giaudrone+2005] Giaudrone, V.; Guerra, M.; Vaccaro, M.; Motz, R.; Marotta, A.: "*A Wrapper & Mediator Prototype for Web Data Warehouses*". In Proc. of the XI Congreso Argentino de Ciencias de la Computación (CACIC'2005), Concordia, Argentine, 2005.

[Gravano+2001] Gravano, L.; Ipeirotis, P. Jagadish, H.V.; Koudas, N. Muthukrishnan, S.; Srivastava, D.: "*Approximate String Joins in a Database (Almost) for Free*". In Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB'2001), Roma, Italy, 2001.

[Graveleau 2005] Graveleau, D.: "*SILURE, mise en œuvre d'un meta-modèle associant traçabilité et qualité des données pour la constitution d'une base de référence multi-sources en veille technologique*". 1st workshop on Data and Knowledge Quality (DKQ'2005), Paris, France, 2005.

[Grigori+2005] Grigori, D.; Peralta, V.; Bouzeghoub, M.: "*Service Retrieval Based on Behavioral Specifications and Quality Requirements*". Accepted paper for the 3rd Int. Conf. on Business Process Management (BPM'2005), Nancy, France, September 2005.

[Gupta+1995] Gupta, A. Mumick, I.: "*Maintenance of Materialized Views: Problems, Techniques, and Applications*". Data Engineering Bulletin, June 1995.

[Gupta 1997] Gupta, H.: "*Selection of Views to Materialize in a Data Warehouse*". Conf. on Database Theory, 1997.

[Hammer+1995] Hammer, J.; Garcia-Molina, H.; Widom, J.; Labio, W.; Zhuge, Y.: "*The Stanford Data Warehousing Project*". IEEE Data Engineering Bulletin, Volume 18, Number 3. June 1995.

[Han+2003] Han, Q.; Venkatasubramanian, N.: "*Addressing Timeliness/Accuracy/Cost Tradeoffs in Information Collection for Dynamic Environments*". In Proc. of the 24th IEEE Int. Real-time Systems Symposium (RTSS'03), Cancun, Mexico, 2003.

[Harinarayan+1996] Harinarayan, V.; Rajaraman, A.; Ullman, J.: "*Implementing Data Cubes Efficiently*". In Proc. of the 1996 ACM Int. Conf. on Management of Data (SIGMOD'96), Montreal, Canada, 1996.

[Helfert+2002] Helfert, M.; Herrmann, C.: "*Proactive Data Quality Management for Data Warehouse Systems*". In Proc. of the Int. Workshop on Design and Management of Data Warehouses, Toronto, Canada, 2002.

[Hillier+1991] Hillier, F.S.; Lieberman, G.J.: "*Introducción a la Investigación de Operaciones*". McGraw-Hill. ISBN 028914-X, 1991.

[Huang+1994] Huang, Y.; Sloan, R.; Wolfson, O.: "*Divergence caching in client-server architectures*". In Proc. of the 3rd Int. Conf. on Parallel and Distributed Information Systems (PDIS 94), pages 131-139. Austin, USA. 1994.

[Hull+1996] Hull, R.; Zhou, G.: "*A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches*". In Proc. of the 1996 ACM Int. Conf. on Management of Data (SIGMOD'96), pages 481-492, Montreal, Canada. 1996.

[Inmon 1996] Inmon, W.: "*Building the Data Warehouse*". John Wiley & Sons Inc., 1996.

[Jankowka 2000] Jankowska, M.A.: "*The need for environmental information quality*". Issues in Science and Technology Librarianship. URL: http://www.library.ucsb.edu/istl/00-spring/article5.html. Last modified in 2000.

[Jermaine+2003] Jermaine, C.: "*Robust Estimation With Sampling and Approximate Pre-Aggregation*". In Proc. of the 29th Int. Conf. on Very Large Data Bases (VLDB'2003), Berlin, Germany, 2003.

[Jarke+1997] Jarke, M.; Vassiliou, Y.: "*Data Warehouse Quality: A Review of the DWQ Project*". In Proc. 2nd Conference on Information Quality (IQ'1997), Cambridge, USA, 1997.

[Jarke+1999] Jarke, M.; Jeusfeld, M.A.; Quix, C.; Vassiliadis, P.: "*Architecture and Quality in Data Warehouses: An Extended Repository Approach*". Information Systems, vol. 24(3), 1999.

[Jeusfeld+1998] Jeusfeld, M. A.; Quix, C.; Jarke, M.: "*Design and Analysis of Quality Information for Data Warehouses*". In Proc. of 17th Int. Conf. on Conceptual Modeling (ER), Singapore, November 16-19, 1998.

[Kieβling+2002] Kieβling, W.; Kôstler, G.: "*Preference SQL - Design, Implementation, Experiences*". In Proc. of the 28th Int. Conf. on Very Large Databases (VLDB'02), Hong Kong, China, 2002.

[Kon+1995] Kon, H.; Madnick, S.; Siegel, M.: "*Good Answers from Bad Data:a Data Management Strategy*". Working paper 3868-95, Sloan School of Management, Massachusetts Institute of Technology, USA, 1995.

[Kostadinov+2004] Kostadinov, D. ; Peralta, V. ; Soukane, A. ; Xue, X.: "*Système adaptatif d'aide à la génération de requêtes de médiation*". Demonstration. In Proc. of the 20èmes Journées de Bases de Données Avancées (BDA'04), Montpellier, France, 2004.

[Kostadinov+2005] Kostadinov, D.; Peralta, V.; Soukane, A.; Xue, X.: "*Intégration de données hétérogènes basée sur la qualité*". XXIII Congrès INFORSID (INFORSID'2005), Grenoble, France, 2005.

[Kostadinov 2006] Kostadinov, D.: "*Personnalisation de l'information et gestion de profils utilisateur*". PhD Thesis, Université de Versailles, France, on going work..

[Klug 1988] Klug, A.: "*On conjunctive queries containing inequalities*". Journal of the ACM, Vol. 35(1):146-160, 1988.

[Labrinidis+2003] Labrinidis, A.; Roussopoulos, N.: "*Balancing Performance and Data Freshness in Web Database Servers*". In Proc. of the 29th Int. Conf. on Very Large Data Bases (VLDB'03), Berlin, Germany, 2003.

[Laboisse 2005] Laboisse, B.: "*BDQS, une approche dans la mesure de la qualité de données d'un CRM : principes de base (les attributs de la qualité), intégration des outils métier de marketing direct dans la mesure*". Séminaire CRM & Qualité des Données, Paris, France, 2005.

[Lee+2000] Lee, M.L.; Ling, T.W.; Lu, H.; Ko, Y.T.: "*Cleansing Data for Mining and Warehousing*". In Proc. of the 10th Int. Conf. on Database and Expert Systems Applications (DEXA '99), Florence, Italy, 1999.

[Levy+1996] Levy, A.; Rajaraman, A.; Ordille, J.: "*Querying Heterogeneous Information Sources Using Source Descriptions*". In Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'1996), 1996.

[Li+2003] Li, D.; Sun, X.: "*Recent Progress in Nonlinear Integer Programming*". Optimization Research Bridge (ORB), Issue 11, September 2003.

[Li+2003a] Li, W.S.; Po, O.; Hsiung, W.P.; Selçuk Candan, K.; Agrawal, D.: "*Freshness-driven adaptive caching for dynamic content Web sites*". Data & Knowledge Engineering (DKE), Vol. 47(2):269-296, 2003.

[Maletic+2000] Maletic, J.I.; Marcus, A.: "*Data Cleansing: Beyond Integrity Analysis*". In Proc. of the Int. Conf. on Information Quality (IQ'2000), Cambridge, USA, 2000.

[Mannino+2004] Mannino, M.; Walter, Z.: "*A Framework for Data Warehouse Refresh Policies*". Technical Report CSIS-2004-001, University of Colorado at Denver, 2004.

[Marotta 2000] Marotta, A.: "*Data Warehouse Design and Maintenance through schema transformations*". Master thesis, Universidad de la República, Uruguay, 2000.

[Marotta+2001] Marotta, A.; Motz, R.; Ruggia, R.: "*Managing Source Schema Evolution in Web Warehouses*". In Proc. of the International Workshop on Information Integration on the Web (WIIW'2001), Rio de Janeiro, Brazil, 2001.

[Marotta 2006] Marotta, A.: "*Managing source quality Changes in a Data Integration System*". Doctoral Consortium of 18th Int. Conf. on Advanced Information Systems Engineering (CAISE'06), Luxembourg, Luxembourg, 2006.

[Mazzi+2005] Mazzi, G.L.; Museux, J.M.; Savio, G.: "*Quality measures for economic indicators*". Statistical Office of the European Communities, Eurostat. ISBN 92-894-8623-6. 2005.

[Mecella+2002] Mecella, M.; Scannapieco, M.; Virgillito, A.; Baldoni, R.; Catarci, T.; Batini, C.: "*Managing Data Quality in Cooperative Information Systems*". In Proc. on the Confederated Int. Conf. DOA, CoopIS and ODBASE (DOA/CoopIS/ODBASE'02), Irvine, USA, 2002.

[Mendling+2006] Mendling, J.; Lassen, K.; Zdun, U.: "*Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages*". In Proc. of Multikonferenz Wirtschaftsinformatik 2006 (MKWI'2006), Band 2, XML4BPM Track, GITO-Verlag, Berlin, Germany, 2006.

[Mihaila+2000] Mihaila, G.; Raschid, L.; Vidal, M.E.: "*Using Quality of Data Metadata for Source Selection and Ranking*". In Proc. of the 3rd Int. Workshop on the Web and Databases (WebDB'2000), Dallas, USA, 2000.

[Missier+2001] Missier, P.; Scannapieco, M.; Batini, C.: "*Cooperative Architectures: Introducing Data Quality*". Technical Report 14-2001, Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza", Roma, Italy, 2001.

[Missier+2003] Missier, P.; Lalk, G.; Verykios, S.; Grillo, F.; Lorusso, T.; Angeletti, P.: "*Improving data quality in practice: a case study in the Italian public administration*". Distributed and Parallel Databases, Vol. 13(2): 135-160, 2003.

[Morey+1982] Morey, R.: "*Estimating and improving the quality of information in MIS*". Communications of the ACM, Vol. 25(5): 337-342, 1982.

[Motro 1995] Motro, A.: "*Management of Uncertainty in database Systems*". Modern Database Systems, ACM Press and Addison-Wesley, ISBN 0-201-59098-0, pages 457-476, 1995.

[Motro+1997] Motro, A.; Rakov, I.: "*Not all answers are equally good: estimating the quality of database answers*". In *Flexible Query-Answering Systems* (T. Andreasen, H. Christiansen, and H.L. Larsen, Eds.), Kluwer Academic Publishers, 1997, Ch. 1, pp. 1–21.

[Motro+1998] Motro, A.; Rakov, I.: "*Estimating the quality of databases*". In Proc of the 3rd Int. Conf on Flexible Query Answering Systems (FQAS'98), Roskilde, Denmark, 1998.

[Moura+2004] Moura, G.; Machado, M.L.: "*AQUAWARE: A Data Quality Support Environment for Data Warehousing*". In Proc. of the 19th Brazilian Symposium on Databases (SBBD'2004), Brasilia, Brazil, 2004.

[Müller+2003] Müller, H.; Freytag, J.C.: "*Problems, Methods, and Challenges in Comprehensive Data Cleansing*". Technical Report, HUB-IB-164, Humboldt University Berlin, Berlin, Germany, 2003.

[Müller+2003a] Müller, H.; Naumann, F.:_ "*Data Quality in Genome Databases*". In Proc. of the. 8th Int. Conf. on Information Quality (IQ 2003), MIT, USA, 2003.

[Naumann+1998] Naumann, F.; Freytag, J.C.; Spiliopoulou, M.: "*Quality Driven Source Selection Using Data Envelope Analysis*". In Proc. of the MIT Conference on Information Quality (IQ'98), Cambridge, USA, 1998.

[Naumann+1999] Naumann, F.; Leser, U.; Freytag, J.C.: "*Quality-driven Integration of Heterogeneous Information Systems*". In Proc. of the 25th Int. Conf. on Very Large Databases (VLDB'99), Scotland, 1999.

[Naumann+2000] Naumann, F.; Rolker, C.: "*Assessment Methods for Information Quality Criteria*". In Proc. of the MIT Conf. on Information Quality (IQ'00), Cambridge, USA, 2000.

[Navarro 2001] Navarro, G.: "*A guided tour to approximate string matching*". ACM Computing Surveys, Vol. 33(1):31-88, 2001.

[Nie+2002] Nie, Z.; Nambiar, U.; Vaddi, S.; Kambhampati, S.: "*Mining Coverage Statistics for Websource Selection in a Mediator*". Technical Report, ASU CSE TR 02-009, 2002.

[Oliveira+2004] Oliveira, P.; Rodrigues, F.; Henriques, P.: "*Limpeza de Dados - Uma Visão Geral*". In Proc. of Data Gadgets Workshop, Malaga, Spain, 2004.

[Oliveira+2005] Oliveira, P.; Rodrigues, F.; Henriques, P.; Galhardas, H.: "*A Taxonomy of Data Quality Problems*". In Proc. of 2nd Int. Workshop on Data and Information Quality (DIQ'2005), Porto, Portugal, 2005.

[Ozsu+1991] Ozsu, M.; Valduriez, P.: "*Principles of Distributed Database Systems*". Prentice-Hall Int. Editors. 1991.

[Parssian+1999] Parssian, A.; Sarkar, S.; Jacob, V.S.: "*Assessing Data Quality for Information Products*". In Proc. of the 20th Int. Conf. on Information Systems (ICIS'1999), Charlotte, USA, 1999.

[Peralta+2004] Peralta, V.; Ruggia, R.; Kedad, Z.; Bouzeghoub, M.: "*A Framework for Data Quality Evaluation in a Data Integration System*". In Proc. of the 19th Brazilian Symposium on Databases (SBBD'2004), Brasilia, Brazil, 2004.

[Peralta+2004a] Peralta, V.; Bouzeghoub, M.: "*On the evaluation of data freshness in data integration systems*". In Proc. of the 20èmes Journées de Bases de Données Avancées (BDA'2004), Montpellier, France, 2004.

[Peralta+2004b] Peralta, V.; Ruggia, R.; Bouzeghoub, M.: "*Analyzing and Evaluating Data Freshness in Data Integration Systems*". Ingénierie de Systèmes d'Information (ISI), vol 9 (5/6) : 145-162, 2004.

[Peralta+2005] Peralta, V.; Bouzeghoub, M.: "*Data Freshness Evaluation in Different Application Scenarios*". In Proc. of the 1st Data and Knowledge Quality Workshop (DKQ'2005), collocated with the 5èmes Journées d'extraction et Gestion des Connaissances (EGC'2005), Paris, France, 2005.

[Peralta 2006] Peralta, V.: "*Evaluating Data Freshness in Web Warehousing Applications*". Technical Report, In.Co., Universidad de la República, Uruguay, 2006.

[Pipino+2002] Pipino, L.L.; Lee, Y.W.; Wang, R.: "*Data Quality Assessment*". Communications of the ACM, vol. 45, No. 4ve, April 2002.

[Pitsoulis+2001] Pitsoulis, L.; Resende, M.: "*Greedy randomized adaptive search procedures*". Handbook of Applied Optimization, P.M. Pardalos and M.G.C. Resende, editors, pp: 168–181, Oxford University Press, 2001.

[Poosala+1997] Poosala, V.; Ioannidis, Y.; Haas, P.; Shekita, E.: "*Selectivity estimation without the attribute value independence assumption*". In Proc. of the 23rd Int. Conf. on Very Large Databases (VLDB'97), Athens, Greece, 1997.

[Quass 1999] Quass, D.: "*A framework for research in data cleaning*" Draft, Brigham Young University, 1999.

[Rahm+2000] Rahm, E.; Do, H. H.: "*Data Cleaning: Problems and Current Approaches*". IEEE Data Engineering Bulletin, Vol. 23(4): 3-13, 2000.

[Rakov 1998] Rakov, I.: "*Data quality and its use for reconciling inconsistencies in multidatabase environment*". PhD Thesis, George Mason University, Virginia, USA, 1998.

[Raman+2001] Raman, V.; Hellerstein, J.: "*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.

[Ramos+2006] Ramos, M.; Settimo, R.: "*Data Quality Evaluation: Herramienta para evaluación y configuración de la calidad en sistemas de integración de datos*". Pre-grade Report, Facultad de Ingeniería, Universidad de la República, Uruguay, 2006.

[Redman 1996] Redman, T.: "*Data Quality for the Information Age*". Artech House, 1996.

[Rouiller 2005] Rouiller, M.J.: "*Una Plataforma para Análisis de Calidad de Datos*". Internal Report, Facultad de Ingeniería, Universidad de la República, Uruguay, 2006.

[Sattler+2000] Sattler, K.U.; Conrad, S.; Saake, G.: "*Adding Conflict Resolution Features to a Query Language for Database Federations*". In Proc. of the 3rd Workshop on Engineering Federated Information Systems (EFIS'2000), Dublin, Ireland, 2000.

[Schurr+2002] Schurr, P.; Chengalur-Smith, I.; Pazer, H.: "Information Quality and Online B2B Relationships After the Purchase". Draft paper. School of Business, University at Albany, USA, 2002.

[Segev+1990] Segev, A.; Weiping, F.: "*Currency-Based Updates to Distributed Materialized Views*". In Proc. of the 6th Int. Conf. on Data Engineering (ICDE'90), Los Angeles, USA, 1990.

[Shanks+1999] Shanks, G.; Corbitt, B.: "*Understanding Data Quality: Social and Cultural Aspects*". In Proc. of the 10th Australasian Conference on Information Systems, Wellington, New Zealand, 1999.

[Shankaranarayan+2003] Shankaranarayan, G.; Ziad, M.; Wang, R.: "*Managing data quality in dynamic decision environments: an information product approach*". Journal of Database Management, Vol. 14(4):14-32, 2003.

[Shin 2003] Shin, B.: "*An exploratory Investigation of System Success Factors in Data Warehousing*". Journal of the Association for Information Systems, Vol. 4: 141-170, 2003.

[Strong+1997] Strong, D.; Lee, Y.; Wang, R.: "*Data Quality in Context*". Communications of the ACM, Vol. 40(5), May 1997.

[Theodoratos+1997] Theodoratos, D.; Sellis, T.: "*Data Warehouse Configuration*". In Proc. of the 23rd Int. Conference on Very Large DataBases (VLDB'1997), Athens, Greece, 1997.

[Theodoratos+1999] Theodoratos, D.; Bouzeghoub, M.: "*Data Currency Quality Factors in Data Warehouse Design*". In Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany. 1999.

[USEPA 2004] U.S. Environment Protection Agency: "*Increase the Availability of Quality Health and Environmental Information*". Disponible en http://www.epa.gov/oei/increase.htm modificado en agosto 2004.

[USNARA 2000] U.S. National Archives & Records Administration: "*The Soundex Indexing System*". URL: http://www.archives.gov/research_room/genealogy/census/soundex.html. Accessed on June 2005.

[van der Aalst+2002] van der Aalst, W.; Hirnschall, A.; Verbeek, H.: "*An Alternative Way to Analyze Workflow Graphs*". In Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering (CAISE'2002), Toronto, Canada, 2002.

[van der Aalst+2003] van der Aalst, W.; Hofstede, A.; Kiepuszewski, B.; Barros, A.: "*Workflow Patterns*". In Distributed and Parallel Databases, Vol 14(1): 5-51, 2003.

[van der Meyden 1992] van der Meyden, R.: "*The Complexity of Querying Indefinite Information*". PhD Thesis, Rutgers University, 1992.

[Vassiliadis+2000] Vassiliadis, P.; Bouzeghoub, M.; Quix, C.: "*Towards Quality-oriented Data Warehouse Usage and Evolution*". Information Systems, Vol. 25(2): 89-115, May 2000.

[Vassiliadis+2001] Vassiliadis, P.; Vagena, Z.; Skiadopoulos, S.; Sellis, T.: "*ARKTOS: towards the modeling, design, control and execution of ETL processes*". Information Systems, Vol 26(8): 537-561, 2001.

[Vila+2006] Vila, D.; Balestra, M.: "*Evolución de Sistemas de Web Warehousing guiado por parámetros de calidad*". Pre-grade project, Facultad de Ingeniería, Universidad de la República, Uruguay, 2006.

[Wand+1996] Wand, Y.; Wang, R.: "*Anchoring Data Quality Dimensions in Ontological Foundations*". Communications of the ACM, Vol. 39(11):86-95, 1996.

[Wang+1996] Wang, R.; Strong, D.: "*Beyond accuracy: What data quality means to data consumers*". Journal on Management of Information Systems, Vol. 12 (4):5-34, 1996.

[Weikum 1999] Weikum, G.: "*Towards guaranteed quality and dependability of information systems*". In Proc. of the Conf. Datenbanksysteme in B*uro, Technik und Wissenschaft, Freiburg, Germany, 1999.

[Widom 1995] Widom, J.: "*Research Problems in Data Warehousing*". In Proc. of the 4th Int. Conf. on Information and Knowledge Management (CIKM'95), Baltimore, USA. 1995.

[Wiederhold 1992] Wiederhold, G.: "*Mediators in the architecture of future information systems*". IEEE Computer, Vol. 25(3):38-49, 1992.

[Wikipedia 2006] Wikipedia. URL: www.wikipedia.org. Last accessed on August 2006

[Xue 2006] Xue, X: "*Automatic Mapping Generation and Adaptation for XML Data Sources*". PhD Thesis, Université de Versailles, France, on going work.

[Yang+1997] Yang, J. Karlapalem, K. Li, Q.: "*Algorithms for materialized view design in data warehousing environment*". In Proc. of the 23rd Int. Conf. on Very Large Databases (VLDB'97), Athens, Greece, 1997.

[Zhuge+1997] Zhuge, Y.; Garcia-Molina, H.; Wiener, J.: "*Multiple View Consistency for Data Warehousing*". In Proc. of the 13th Int. Conf. on Data Engineering (ICDE'97), Birmingham, UK.1997.

[Zhu+2002] Zhu, Y.; Buchmann, A.: "*Evaluating and Selecting Web Sources as External Information Resources of a Data Warehouse*". In Proc. of the 3rd Int. Conf. on Web Information Systems Engineering (WISE'02), Singapore, 2002.

[Ziemann+2005] Ziemann, J.; Mendling, J.: "*EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach*". In Proc. of the 7th Int. Conf. on Modern Information Technology in the Innovation Processes of the Industrial Enterprises (MITIP 2005), Genova, Italy, 2005.

Annex A. Design of the DQE tool

This annex presents the data model of DQE and its storage in the metabase.

1. Data model

In this section we present a simplified version of the data model of DQE, focusing in the most important classes and omitting auxiliary ones.

Framework components and sessions

The framework is composed of a set of quality graphs, a set of data sources, a set of data targets, a set of properties and a set of algorithms. The formal definitions of the framework and its components can be found in Sub-section 2.1 of Chapter 3. Sessions contains sub-sets of such components. Data sources, data targets, properties and algorithms can be contained in several sessions but quality graphs can be contained in a unique session. Figure A.1 shows framework and sessions components.

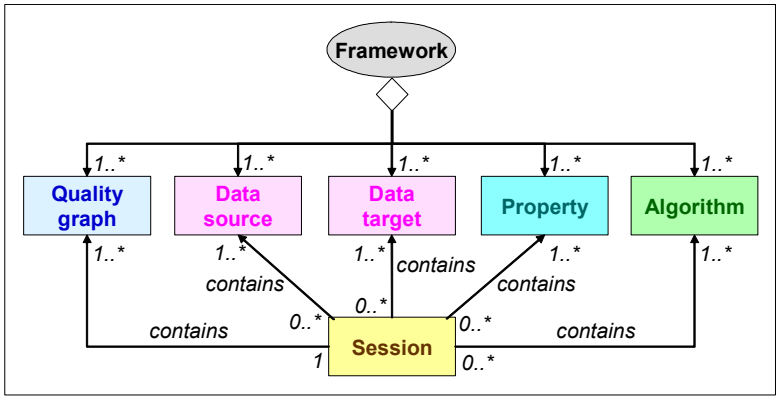

Figure A.1 – Conceptual representation of global components and session components

The remaining of the section describes the components and their inter-relations*.

Quality graphs, data sources and data targets

Quality graphs are composed of nodes and edges. Nodes can be of three types: source nodes, target nodes, or activity nodes. The two former reference the corresponding data sources and data targets respectively. Edges relate two nodes. In order to reuse Java graph libraries, we choose to represent mono-edged graphs, i.e. graphs that have a unique edge between a pair of nodes. Edges can be of three types: data edges, control edges and mixed edges; the latter represents the existence of a data edge and a control edge between the nodes. Figure A.2 illustrates the representation of quality graphs, data sources and data targets.

* In the remaining figures, we color the framework components with the same colors they appear in Figure A.1 in order to quickly identify their sub-components and the relationships with other framework components.

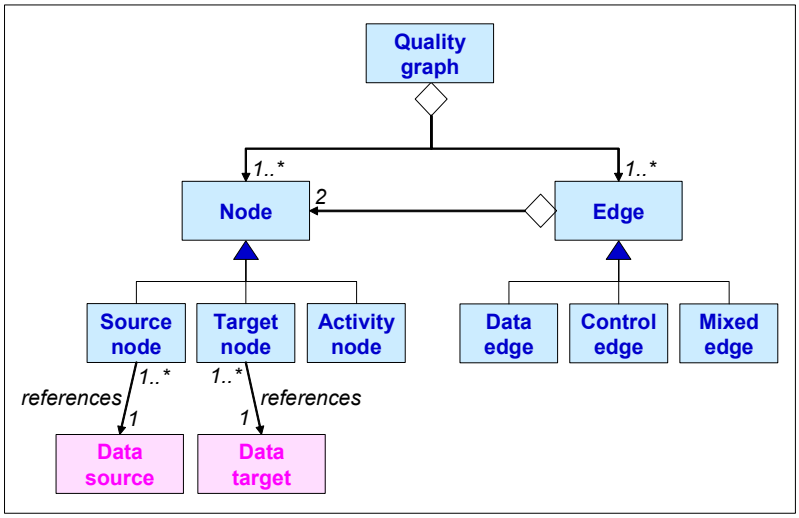

Figure A.2 – Conceptual representation of quality graphs, data sources and data targets

Properties

Properties can be of two types: features and measures. In the case of a measure, we also model the hierarchy *quality dimension* → *quality factor* → *quality metric*, as shown in Figure A.3.

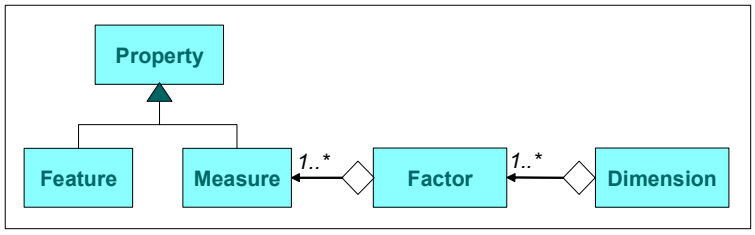

Figure A.3 – Conceptual representation of properties

The association of properties to nodes and edges is done indirectly, via groups of nodes and groups of edges. In other words, nodes and edges are grouped according to the properties they should have and properties are associated to the groups. This allows associating the relevant properties once and not for each node/edge. Figure A.4 shows the grouping of some nodes in three node groups and the association of properties to such groups; for example, as $Node_1$ belongs to $Group_1$ it has indirectly associated two properties: $Prop_1$ and $Prop_2$.

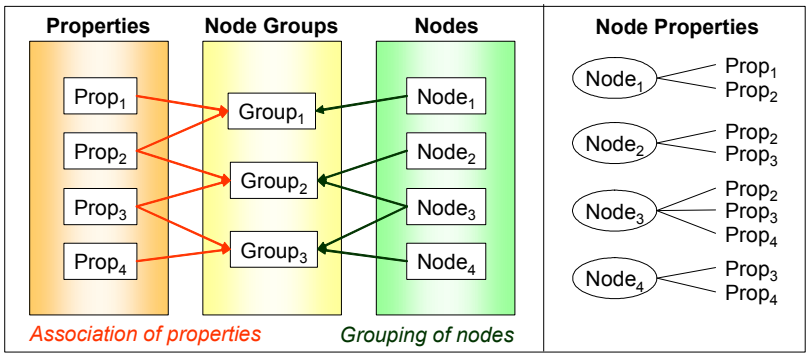

Figure A.4 – Association of properties to nodes

There are two types of associations: global and local. In the former, properties are globally associated to groups and therefore they are valid for all quality graphs whose nodes/edges belong to such groups. In the latter, the association is local to a quality graph, i.e. only the nodes/edges of the graph belonging to such groups are

affected. Nodes and edges are labeled with property values (when they belong to a group that has associated the property). For generality purposes, some property values can label the whole quality graph (e.g. the label *DISadministrator='VP'*). Figure A.5a illustrates the grouping of nodes and edges and their association of property values. As data sources and data targets can also have property values, source nodes and target nodes inherit the property values of the referenced sources and targets respectively, as illustrated in Figure A.5b.

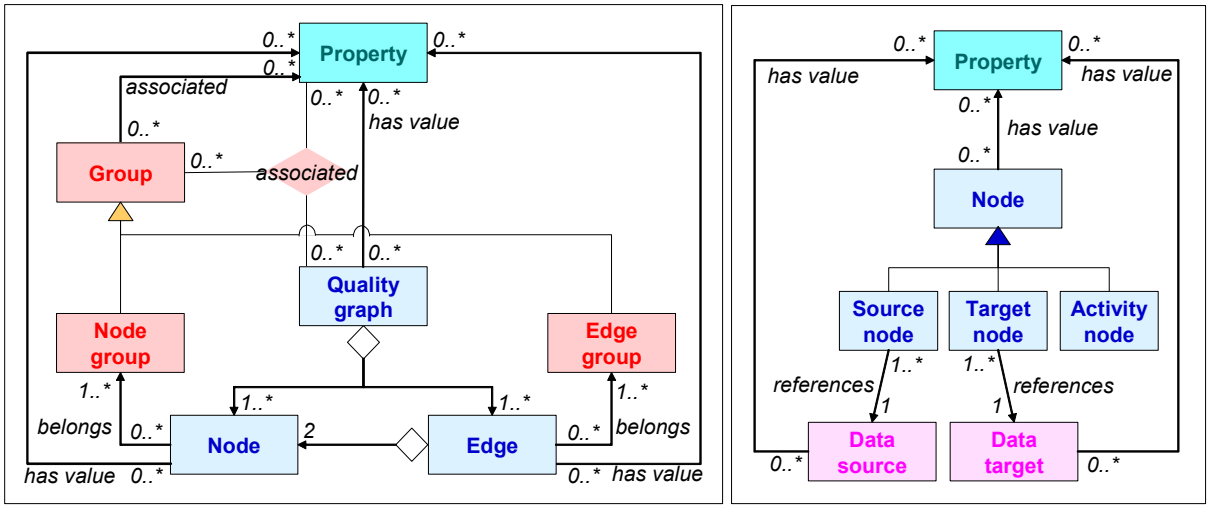

Figure A.5 – Conceptual representation of the association of property values: (a) grouping of nodes and edges, and (b) inheritance of source and target properties

Algorithms

Algorithms have associated pairs <group, property> indicating the groups that must be labelled with certain properties as precondition for executing and the groups that will be labelled with certain properties as postcondition of the execution. They also indicate the quality property that they calculate, as shown in Figure A.6.

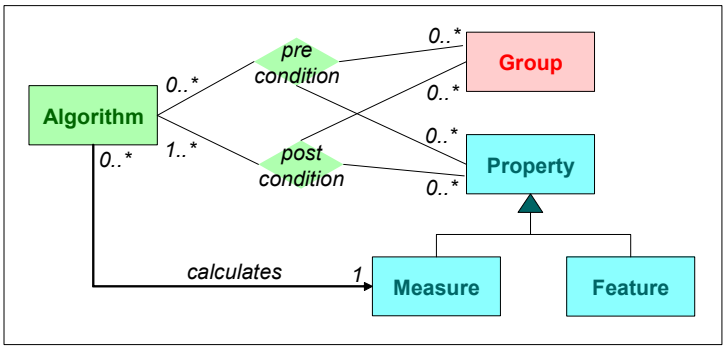

Figure A.6 – Conceptual representation of algorithms

2. Metabase

The metabase provides the persistency of the data model. The relational schema of the metabase is shown in Figure A.7; tables are colored according to the framework components that they represent, namely: ■ catalogues of data sources and data targets, ■ catalogue of quality graphs, ■ catalogue of properties, ■ catalogue of algorithms, ■ sessions, ■ grouping and ■ property values; bold attributes represent keys; continuous lines among relations represent foreign key dependencies while dashed arrows represent optional references (treated as foreign key depending on the value of other attributes).

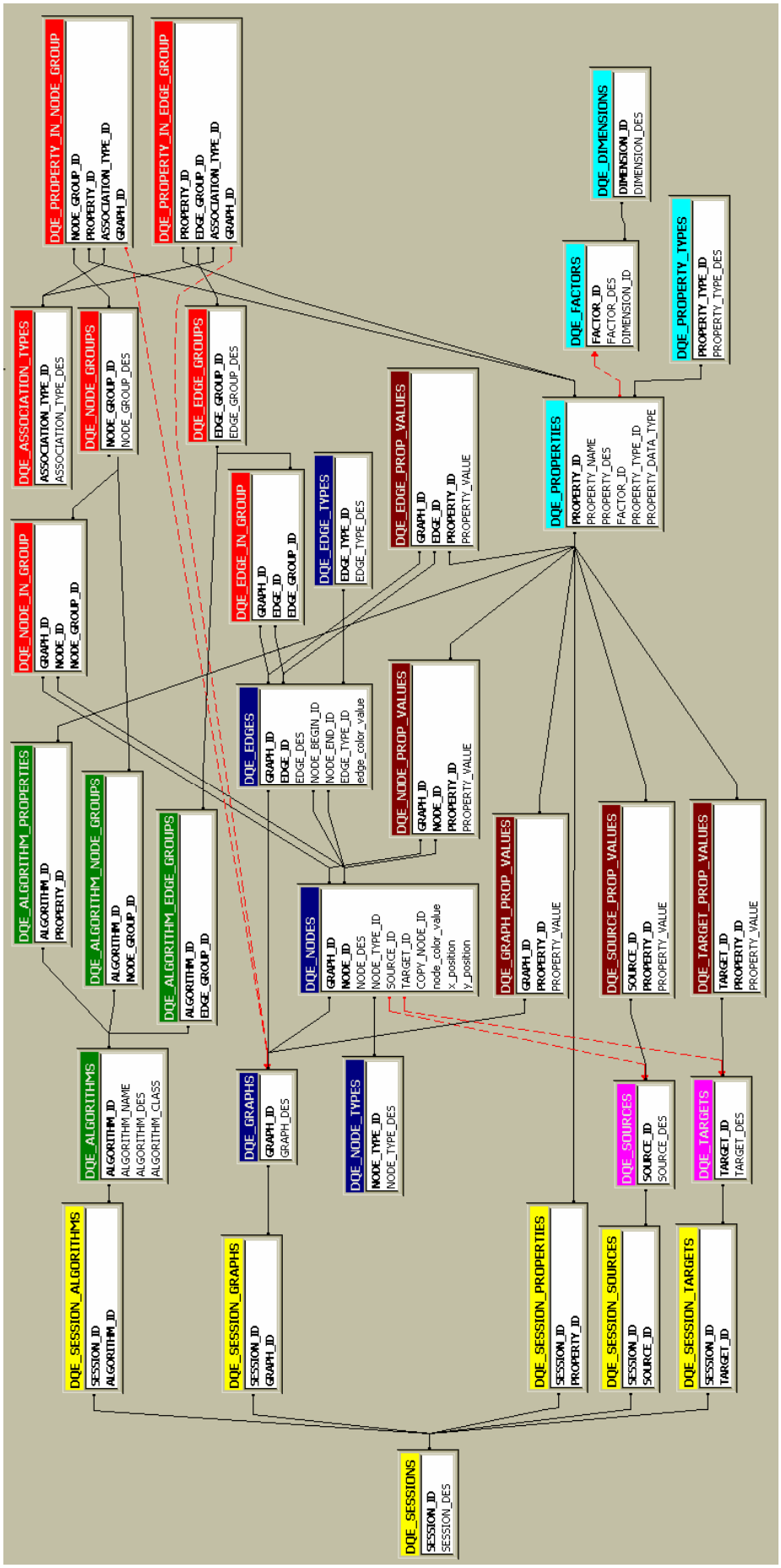

Figure A.7 – Metabase schema

Annex B. Instantiation of the Freshness Evaluation Algorithm

This annex presents the overloaded functions that instantiate the freshness evaluation algorithm in two application scenarios: a Mediation application and a Web Warehousing application.

1. Mediation application scenario

In this section we recall the relevant properties of the mediation application scenario and we present the pseudocodes of the corresponding overloaded functions.

Four freshness evaluation algorithms where proposed:

- *TimelinessEvaluation1*: It is the algorithm used in virtual contexts or when user expectations range several months. Processing costs and inter-process delays (including those caused by data materialization) are neglected. Source data actual timeliness is considered.
- *TimelinessEvaluation2*: It is the algorithm used in materialization contexts when user expectations range several days or weeks. Inter-process delays due to data materialization are considered, as well as source data actual timeliness. Processing costs and other inter-process delays are neglected.
- *CurrencyEvaluation1*: It is the algorithm used in virtual contexts. Processing costs are estimated from the number of input tuples and inter-process delays are neglected. Source data actual currency is neglected.
- CurrencyEvaluation2: It is the algorithm used in materialized contexts. Processing costs and inter-process delays among operation nodes are irrelevant compared to refreshment periods, hence, the unique property value that is consider is the inter-process delay cause by data materialization.

Table B.1 recalls the calculation of properties, which was discussed in Sub-section 3.1.3 of Chapter 5 (Table B.1 is adapted from Table 5.1).

	Data timeliness 1	**Data timeliness 2**	**Data currency 1**	**Data currency 2**
Processing cost (A) one predecessor B	Neglect	Neglect	*Tuples*(B) / *Capacity*(A)	Neglect
Processing cost (A) two predecessors B_1 and B_2	Neglect	Neglect	*Tuples*(B_1) * *Tuples*(B_2) / *Capacity*(A)	Neglect
Inter-process delay (between operators)	Neglect	Neglect	Neglect	Neglect
Inter-process delay (between the last operator A and the interface I)	Neglect	*Refreshment period* (A)	*Refreshment period* (A)	*Refreshment period* (A)
Source data actual freshness (S), successor A	*Update period* (S)	*Update period* (S)	Neglect	Neglect
Combination of input values	Maximum of input values	Maximum of input values	Maximum of input values	Maximum of input values

Table B.1 – Calculation of property values with different types of estimation

Algorithm B.1 shows the pseudocodes of the overloaded functions for the second algorithm (timeliness 2 in Table B.1). The *getSourceActualFreshness* function returns the source update period. The *getProcessingCost* function returns zero because processing cost is neglected. The *getInterProcessDelay* function returns zero for all edges except that incoming the mediator interface, which is calculated as the refreshment period. The *combineActualFreshness* function returns the maximum of input values.

For the first algorithm (timeliness 1), the *getInterProcessDelay* function returns zero; the other functions are reused from Algorithm B.1.

For the fourth algorithm (currency 2), the *getSourceActualFreshness* function returns zero; the other functions are reused form Algorithm B.1.

For the third algorithm (currency 1), the *getProcessingCost* function is calculated from the tuples and capacity properties, as shown in Algorithm B.2; the other functions are reused form Algorithm B.1.

```
FUNCTION getSourceActualFreshness (G: QualityGraph, A: Node) RETURNS INTEGER
   INTEGER value = G.getPropertyValue(A,"UpdatePeriod");
   RETURN value;
END
```

```
FUNCTION getProcessingCost (G: QualityGraph, A: Node) RETURNS INTEGER
   RETURN 0;
END
```

```
FUNCTION getInterProcessDelay (G: QualityGraph, e: Edge) RETURNS INTEGER
   NODE A= e.source, B= e.target;
   IF G.belongsToGroup(B,"Interface") THEN
      INTEGER value = G.getPropertyValue(A,"RefreshPeriod");
      RETURN value;
   ELSE
      RETURN 0;
END
```

```
FUNCTION combineActualFreshness (G: QualityGraph, valList: HushTable) RETURNS INTEGER
   INTEGER aux= 0;
   FOR EACH <e, value> in valList DO
      IF value > aux
         aux = value;
   ENDFOR
   RETURN aux;
END
```

Algorithm B.1 – Overloading of functions for the second algorithm (timeliness 2)

```
FUNCTION getProcessingCost (G: QualityGraph, A: Node) RETURNS INTEGER
   INTEGER tuples;
   INTEGER capacity = G.getPropertyValue(A,"Capacity");
   LIST OF NODE nList = G.getPredecessors (A);
   IF (nList.getSize == 1) THEN
      NODE B= nList.getFirst();
      tuples = G.getPropertyValue(B,"Tuples");
      RETURN tuples / capacity;
   ELSE
      NODE B1= nList.getFirst();
      tuples = G.getPropertyValue(B1,"Tuples");
      NODE B2= nList.getLast();
      tuples = tuples * G.getPropertyValue(B2,"Tuples");
      RETURN tuples / capacity;
END
```

Algorithm B.2 – Overloading of the getProcessingCost function for the third algorithm (currency 1)

2. Web warehousing application scenario

In this section we recall the relevant properties of the web warehousing application scenario and we present the pseudocodes of the corresponding overloaded functions.

Table B.2 recalls the calculation of properties, which was discussed in Sub-section 3.2.3 of Chapter 5 (Table B.2 is adapted from Table 5.2 and Table 5.3).

	Precise value	**Average case**	**Worst case**
Processing cost (A)	Neglect	Neglect	Neglect
Inter-process delay (A,B), B in $\{i_1, i_2, t_1, t_2, t_4, t_5\}$	Neglect	Neglect	Neglect
Inter-process delay (A,B), B in $\{m_1, m_2, t_3, v_1, v_2, v_3, v_4, v_5\}$	*Last execution time* (B) – *Last execution time* (A)	Average in statistics of: *Execution time* (B) – *Execution time* (A)	Maximum in statistics of: *Execution time* (B) – *Execution time* (A)
Source data actual currency (S)	Neglect	Neglect	Neglect
Source data actual timeliness (S), push	Neglect	Neglect	Neglect
Source data actual timeliness (S), periodic pull, wrapper W	*Last execution time* (W) – *Last change detection* + *Pull period* (W)	*Pull period* (W) + Average in statistics of: *Execution time* (W) – *Change detection time*	*Pull period* (W) + Maximum in statistics of: *Execution time* (W) – *Change detection time*
Combination of input values (different data volatility)	Input value of most volatile input		
Combination of input values (equal data volatility)	Average of input values weighted with *Data volume*	Average in statistics of: average of input values weighted with *Data volume*	Maximum (input values)

Table B.2 – Calculation of property values with different types of estimation

Algorithm B.3 shows the pseudocodes of the overloaded functions for calculating timeliness with the precise estimation strategy, according to Table B.2. The *getSourceActualFreshness* function returns different values depending on the source capabilities, namely it returns: (i) zero for sources that can announce changes and (ii) the difference between wrapper execution time and last change detection time plus the pull period of the wrapper for the other sources. The *getProcessingCost* function returns zero because processing cost is neglected. The *getInterProcessDelay* function returns different values for the different types of edges, namely it returns: (i) zero when the successor activity belongs to the *NoDelay* group (i.e. activities i_1, i_2, t_1, t_2, t_4 and t_5) and (ii) the difference of execution times for the other activities. The *combineActualFreshness* function traverses the list of input values keeping the total (sum of input values multiplied by data volumes) and volume (sum of data volumes) variables, which allows the calculation of the weighted average at the end. When a more volatile predecessor is found, such values are reinitialized. Note that some property values are obtained from the graph (labels of nodes and edges) but other ones are read in a log.

For calculating currency, the *getSourceActualFreshness* function returns zero, and the other functions are reused from Algorithm B.3.

The pseudocodes of the overloaded functions for the average and worst case estimation strategies are analogous. The difference is that the log methods *getAverage* or *getMaximum* are invoked instead of the *getLast* method (highlighted in red in Algorithm B.3).

```
FUNCTION getSourceActualFreshness (G: QualityGraph, A: Node) RETURNS INTEGER
   INTEGER announce= G.getPropertyValue(A,"AnnounceChanges")
   IF announce = TRUE THEN
      RETURN 0;
   ELSE
      NODE W= G.getSuccessors(A)  /*the wrapper*/
      LOG log = G.getLog("Execution-ChangeDetection");
      INTEGER time = log.getLast(A,W);
      INTEGER pull= G.getPropertyValue(W,"PullPeriod");
      RETURN time + pull;
END
```

```
FUNCTION getProcessingCost (G: QualityGraph, A: Node) RETURNS INTEGER
   RETURN 0;
END
```

```
FUNCTION getInterProcessDelay (G: QualityGraph, e: Edge) RETURNS INTEGER
   NODE A= e.source, B= e.target;
   INTEGER time;
   IF G.isSource(A) or G.isTarget(B) THEN
      RETURN 0;
   ELSE IF G.belongsToGroup(B,"NoDelay") THEN
      RETURN 0;
   ELSE
      LOG log = G.getLog("Execution-Execution");
      INTEGER time = log.getLast(A,B);
      RETURN time;
END
```

```
FUNCTION combineActualFreshness (G: QualityGraph, valList: HushTable) RETURNS INTEGER
   INTEGER maxVolatility= 0, volume= 0, total= 0, volatility, aux;
   FOR EACH <e, value> in valList DO
      NODE A= e.source;
      volatility = G.getPropertyValue(e,"DataVolatility");
      LOG log = G.getLog("DataVolumne");
      INTEGER aux = log.getLast(A);
      IF volatility > maxVolatility THEN
         maxVolatility = volatility;
         volume = aux;
         total = aux * value;
      ELSE IF volatility = maxVolatility THEN
         volume = volume + aux;
         total = total + aux * value;
   ENDFOR
   IF volume = 0 THEN RETURN 0;
   ELSE RETURN total / volume;
END
```

Algorithm B.3 – Overloading of functions for timeliness and precise estimation strategy