Analyzing and Evaluating Data Freshness in Data Integration Systems

Verónika Peralta* — Raúl Ruggia — Mokrane Bouzeghoub***

** Laboratoire PRISM, Université de Versailles*
45, avenue des Etats-Unis
78035, Versailles cedex, FRANCE
Veronika.Peralta@prism.uvsq.fr
Mokrane.Bouzeghoub@prism.uvsq.fr

*** Instituto de Computación, Universidad de la República*
Julio Herrera y Reisig 565, 5to piso
11300, Montevideo, URUGUAY
ruggia@fing.edu.uy

ABSTRACT: *Data freshness has been identified as one of the most important data quality attributes in information systems. This importance increases particularly in the context of systems composed of a large set of autonomous data sources where integrating data having different freshness may lead to semantic problems. This paper addresses the problem of evaluating data freshness in a data integration system and presents a taxonomy to classify different scenarios where data freshness can be evaluated. We propose a framework for modeling the data integration system and performing freshness evaluation and we illustrate the approach for a particular scenario.*

RÉSUMÉ. *La fraîcheur des données est l'un des facteurs de qualité les plus importants dans les systèmes d'information. Cette importance est accrue dans le contexte des systèmes composés d'un grand nombre de sources de données autonomes, où l'intégration des données caractérisées par des fraîcheurs différentes peut mener à des problèmes sémantiques. Cet article adresse le problème d'évaluer la fraîcheur de données dans un système d'intégration de données et présente une taxonomie pour classifier différents scénarios où la fraîcheur de données peut être évaluée. Nous proposons un cadre général pour modeler le système d'intégration de données et réaliser l'évaluation de la fraîcheur et nous illustrons l'approche pour un scénario particulier.*

KEY WORDS: *Data Freshness, Quality evaluation, Quality metrics.*

MOTS-CLÉS: *Fraîcheur des données, Evaluation de la qualité, Métriques de la qualité*

1. Introduction

Data freshness has been identified as one of the most important attributes of data quality for data consumers (Shin 2003). Some surveys and empirical studies have proved that data freshness is linked to information system success (Wang et al., 1996; Mannino et al., 2004; Shin 2003). Specifically, the increasing need to access to information that is available in several data sources introduces the problem of choosing between alternative data providers and of combining data having different freshness values. This paper presents an analysis of data freshness within the context of a Data Integration System (DIS).

A DIS is an information system that integrates data from different independent data sources and provides the users with a uniform access to the data by the mean of a global model. This is sketched in Figure 1. Examples of DIS are: Mediation systems, Data Warehousing systems, Federations of databases and Web Portals.

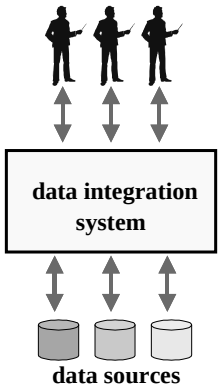


Figure 1 - Data Integration Systems

Several freshness definitions have been proposed for different types of DIS. The traditional freshness definition, called *currency* (Segev et al., 1990), is related to view consistency when materializing data and describes how *stale* is data with respect to the sources. Currency captures the gap between the extraction of data from the sources and its delivery to the users. Recent proposals incorporate another notion of freshness, called *timeliness* (Wang et al., 1996), which describes how *old* is data. Timeliness captures how often data changes or how often new data is created in a source. Therefore, freshness represents a family of quality factors, each one best suiting a particular problem or system.

Although data freshness has been largely studied (Segev et al., 1990; Zhuge et al., 1997; Cho et al., 2000; Li et al., 2003), several problems still remain either unsolved or insufficiently treated: specification of freshness expectations,

acquisition of source freshness, auditing freshness of existing systems, providing support to quality-driven engineering and linking with other quality factors such as performance and accuracy (Bouzeghoub et al., 2003). Among these problems, we address the problem of evaluating data freshness and deciding if user quality expectations can be achieved. An auditing tool should take as input some metadata describing the DIS, sources and query classes, as well as measures of the actual freshness of source data and user freshness requirements. The tool should return a measure of the freshness of the data returned to the user.

In this paper we analyze the dimensions that have impact in the freshness of data, namely, the nature of the data, the system features and the synchronization policies of the underlying management system, and we organize them in a taxonomy for classifying the scenarios where data freshness can be evaluated. We propose a framework that models DIS integration processes as a workflow of calculation activities and performs the evaluation based on the workflow representation. We illustrate the approach using the framework in a particular scenario.

The rest of the document is organized as follows: Section 2 analyzes the dimensions involved in freshness evaluation and presents the taxonomy. Section 3 describes the framework, which is used in section 4 to evaluate data freshness in a particular scenario. Finally, section 5 concludes with our general remarks.

2. Data Freshness

This section describes freshness metrics and techniques and analyzes some dimensions that impact the evaluation and enforcement of data freshness. We also present a taxonomy that summarizes this discussion and enables the classification of application scenarios.

2.1. Overview of Metrics

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. Table 1 describes the metrics proposed in the literature for measuring data freshness, classified by factor. A larger description of factors and metrics can be found in (Bouzeghoub et al., 2004).

These factors and metrics have been used in different systems and different application contexts, among which:

– *View Materialization*: In the context of view materialization, a view is consistent if its state reflects an actual source state at some “recent time” (Zhuge et al., 1997), so the goal is assuring a certain degree of data *currency*. The view maintenance problem consists of updating a materialized view in response to

Factor	Metric	Definition
Currency	Currency	The time elapsed since data was extracted from the source (The difference between query time and extraction time) (Segev et al., 1990; Theodoratos et al., 1999).
	Obsolescence	The number of updates transactions/operations to a source since data extraction time (Gal 1999).
	Freshness rate	The percentage of tuples in the view that are up-to-date (have not been updated since extraction time) (Cho et al., 2000; Labrindis et al., 2003).
Timeliness	Timeliness	The time elapsed from the last update to a source (the difference between query time and last update time) (Naumann et al., 1999; Gertz et al., 2004).

Table 1 – Freshness factors and metrics

changes arisen at source data. Most of the work concentrates in assuring data warehouse consistency for different types of views and refreshment strategies (Gupta et al., 1995). Another key problem has been the selection of a set of views to materialize in order to optimize the query evaluation and/or the maintenance cost (Theodoratos et al., 1997; Baralis et al., 1997). In this context, freshness is implicitly considered when defining the update propagation processes.

– *Caching*: In caching systems, data is considered fresh when it is identical to data in the sources, so freshness is represented by the *currency factor*. An important problem is defining the refreshment policies in order to keep cache data up-to-date. Traditional cache proposals estimate the *time-to-live* (TTL) of an object to check whether this object is valid or not and when to get it from its source. In (Cho et al., 2000), they study the synchronization policies for cache refreshment and experimentally verify their behavior. In (Li et al., 2003), the focus is in the fine-tuning of the caching policy, balancing response time and invalidation cycles for assuring data currency. In (Bright et al., 2002), they propose the use of *latency-recency* profiles to adapt caching algorithms to user currency requirements, accessing the remote site only when the expected currency is not achieved.

– *Virtual Systems*: In pure virtual systems, e.g. classical mediation systems, *currency* is implicitly studied when dealing with response time. A key problem is query optimization, but freshness is not the goal. 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 et al., 1999), they study how to propagate a set of quality factors from several heterogeneous sources to the mediator.

2.2. Dimensions for Freshness Analysis

This section analyzes some dimensions that impact the analysis and enforcement of data freshness: nature of data, system features and synchronization policies.

2.2.1. Nature of Data

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

– *Stable data*: It is data that is improbable to change. 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. The concept of "low frequency" is user dependent.

– *Frequently-changing data*: It is data that has intensive change, such as real-time traffic information, temperature sensor measures and sales quantities.

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. 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.

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.

2.2.2. System Features

The freshness of the data returned to the user depends on the freshness of the extracted data but also on the processes that extract, integrate and deliver this data. These processes are very important because they can introduce additional delays. These delays can 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 can 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

types of techniques: virtual, caching and materialization techniques. 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 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 almost using the materialized data. Examples are data warehousing systems and web portals that support materialization.

Virtual techniques query sources and 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.

2.2.3. Synchronization Policies

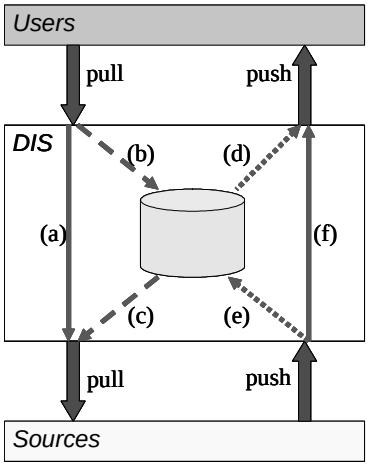
The way the DIS is implemented influences the freshness of the data delivered to the users. Particularly, the synchronization between the sources, the DIS and the users has impact in data freshness because it introduces delays. For example, a DIS that synchronizes updates each end of the day may provide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

According to the interaction between 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. According to the interaction between 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. The notifications can be sent by active agents or can be determined by polling, driven by temporal or non-temporal events.

Combining the previous interactions between users, DIS and data sources leads to six possible configurations which are shown in Figure 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.

Asynchronous policies introduce delays. The refreshment 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.



Configurations are named with the user-DIS policy followed by the DIS-source policy,
(/) represent asynchronisms and (-) synchronisms.

Synchronous configurations:
- pull-pull: arrow (a)
- push-push: arrow (f)

Asynchronous configurations:
- pull / pull: arrows (b) and (c)
- pull / push: arrows (b) and (e)
- push / push: arrows (d) and (e)
- push / pull: arrows (d) and (c)

Figure 2 – Synchronization policies

We organize the previously described dimensions into a taxonomy, which can be used to classify the different scenarios where data freshness can be analyzed. An example of scenario can be a DIS that integrates frequently changing data, using caching techniques and pull/pull policies. The taxonomy is useful because the technical problems to solve for each scenario 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.

There is a correlation between system features 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. However, a priori, all combinations of natures of data and system features are possible, i.e. virtual, caching or materialization techniques (with their valid combinations of synchronization

policies) can be built to query different types of data. In addition, users of these systems can be interested in both freshness factors. 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, obsolesce is most studied in caching systems.

Next section describes a framework for data freshness evaluation that models the DIS and allows expressing the properties of each scenario.

3. A Framework for Data Freshness Evaluation

In this section we present a framework for data freshness evaluation in the context of a DIS. The framework models the DIS processes and properties and evaluates the freshness of the data returned to the user. Data freshness is evaluated through the application of quality evaluation algorithms that embed the properties of the scenarios.

The proposed framework consists of a set of available data sources, classes of user queries, the DIS integration process, a set of properties describing system features and quality measures and a set of quality evaluation algorithms. The following subsections describe the framework and its components.

3.1. Modeling the Data Integration System

The DIS can be viewed as a workflow in which the activities perform the different tasks that extract, transform 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. The data produced by an activity can be immediately consumed by other activities or it can be materialized for being queried later. Note that this notion of activity can represent processes of different complexity; from simple SQL operations to complex transformation procedures that can execute autonomously.

Figure 3 sketches the dataflow representation of a DIS process. On the bottom diagram there are remote sources (Si). On the middle diagram there are the different activities (Ai) whose inputs are source data. The arrows indicate that the output node uses the data returned by the input node. The activities that directly take input data from sources are the wrappers that perform the data extraction from sources. The other activities take input data, directly or indirectly, from wrappers. On the top diagram there are the user query classes (Qi) representing families of queries that can be solved using the data produced by activities.

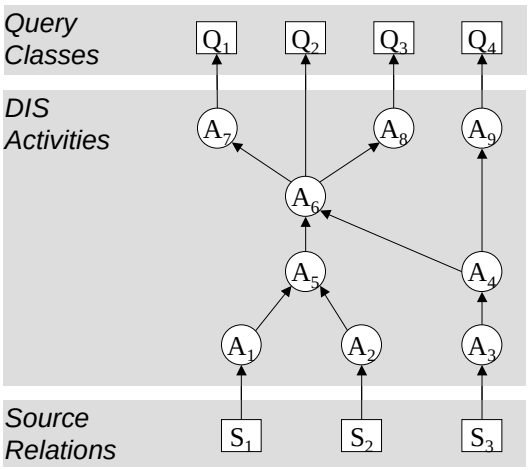


Figure 3 – A dataflow representation of a DIS

Formally, we represent the DIS dataflow by means of a directed acyclic graph (dag) that describes the involved activities, their inputs and outputs. The dag shows the data paths from sources to user queries within the different activities.

Definition 1. A *calculation dag* is a dag G = <V, E> defined as follows: The nodes in V are of three types: *source nodes* (with no input edges) that represent the sources, *target nodes* (with no output edges) that represent query classes and *activity nodes* (with both input and output edges) that represent the different activities that calculate the set of target nodes from the source nodes. The edges in E represent that a node is calculated from another (the data flows in the sense of the arrow). □

Example 1. Consider the DIS of Figure 3 which provides meteorological information. There are three sources: S_1 with real time satellite meteorological predictions, S_2 with official meteorological predictions from a french dissemination database and S_3 with information taken by climatic sensors. The goal of the system is to provide meteorological information to solve four classes of queries: Q_1 (historical information about climate alerts), Q_2 (detailed data comparing predictions), Q_3 (aggregated data about predictions) and Q_4 (aggregated data about climate measurements). The DIS is composed of nine activities that process the information performing the data extraction, integration and aggregation. The activities A_1, A_2 and A_3 are the wrappers, while activity A_4 filters information keeping only the data about the metropolitan regions of France. Activity A_5 integrates information extracted from sources S_1 and S_2, adding comparison indicators and checking some integrity constraints. Activity A_6 joins information produced by A_4 and A_5 and materializes the result. Activity A_7 aggregates information keeping historical materialized data about drastic changes in climate

alerts. Activity A_8 also performs aggregations. Finally, activity A_9 reorganizes input information ordering it by region. □

3.2. Modeling Properties

In this section we describe the properties that express data and system features and support the calculation of data freshness. To carry out data freshness evaluation we firstly need to identify which freshness factor to evaluate, depending on user applications. This implies the selection of metrics, the determination of the relevant properties and the implementation of evaluation algorithms that measure, estimate or bound such quality factor.

In order to calculate quality values corresponding to the selected factor, the algorithms need input information describing system properties such as, for example, the time an activity needs to execute or a descriptor stating if an activity materializes data or not. These properties can be of two types: (i) *descriptions*, indicating some feature of the system (costs, delays, policies, strategies, constraints, etc.), or (ii) *measures*, indicating a quality value corresponding to a quality factor, which can be an actual value acquired from a source, a calculated value obtained executing an evaluation algorithm or an expected value indicating the user desired value for the quality factor. We define a property as follows:

Definition 2. A *property* is a 3-uple <Name, Metric, DataType> where the *Name* is a String that identifies the property, *Metric* is a description of the measurement semantics and units and *DataType* describes the domain of the property values. □

The freshness of the data delivered to the user depends on source data freshness but also on the execution delay of the system, which is the amount of time from data extraction to data delivery. This length of time is influenced not only by the processing cost of each activity (the time the activity needs for executing) but also for the delays that can exist between the executions of consecutive activities. We briefly describe such properties, as well as user expectations:

– *Processing cost*: It is the amount of time, in the worst case, that an activity needs for reading input data, executing and building result data.

– *Synchronization delay*: It is the amount of time passed between the executions of two consecutive activities.

– *Actual Freshness*: It is a measure of the freshness of data in a source, which can be provided by the source or can be estimated or bounded by the system.

– *Expected Freshness*: It is the desired freshness for the data returned by queries. It measures the extent to which the freshness of the data is appropriate for the task on hand (Wang et al., 1996), and is also related to *data volatility* that identifies the time interval in which data remain valid (Ballow et al., 1998).

The relevance of these properties depends on its magnitude compared to freshness requirements, but it also depends on the scenario. For example, the materialization of data and the use of different policies to refresh such data may imply important *synchronization delays* while in virtual systems these delays can be negligible. Other properties can be considered for specific scenarios. The selection of adequate properties for each scenario also depends on the quality factors. Particularly, considering the source actual freshness is only relevant when evaluating *timeliness* because *currency* is measured with respect to source data.

A property can be related to some nodes or edges of the calculation dag. For example, we can *associate actual* freshness to source nodes, *expected freshness* to target nodes and *synchronization delays* to edges. Figure 4 shows the calculation dag of Figure 3 labeled with some properties. The following definition formalizes the association of properties to the calculation dag.

Definition 3. A *labeled calculation dag* is a calculation dag whose nodes and edges have associated a set of property values:

$$G = \langle V, E, P, propvalue \rangle$$

where:

- V and E are sets of nodes and edges.
- P is a set of properties.
- propvalue is a partial function that assigns a value of a property to a node or edge of the dag. It is defined as: $propvalue: (V \cup E) \times \{p / p \in P\} \rightarrow p.DataType$ □

As a labeled calculation dag describes the DIS integration process and its properties, it contains the input information needed by the evaluation algorithms.

The quality evaluation is performed by evaluation algorithms that take as input a labeled calculation dag, calculate the quality values corresponding to a quality factor and return a calculation dag with an additional property (corresponding to the evaluated quality factor). The calculation dag must be labeled with certain properties in order to execute a certain algorithm. For example, the activity nodes of a calculation dag must be labeled with their processing cost in order to execute certain algorithm that evaluates data currency in certain scenario.

The framework does not constraint the way the algorithms can be implemented. For example, an algorithm can evaluate freshness in a bottom-up way, that is, calculating the freshness of the data produced by a node in terms of the freshness of the data coming from input nodes and adding processing costs and synchronization delays. Another algorithm can evaluate the freshness of target nodes in a top-down way, starting at target nodes and adding costs and delays of predecessor nodes until arriving to a source node. The proposed DAG representation facilitates the implementation because it enables to use graph primitives (getPredecessors,

getSuccessors, getProperties, etc.) and traversal methods (findShortestPath, depthFirstSearch, etc.).

In the next section we present an example of how the evaluation algorithms can be implemented.

4. Using the Framework

In this section we show the application of the framework for a particular scenario. We firstly determine the set of properties that impact on the evaluation of data freshness for this scenario and we develop an evaluation algorithm that considers such properties. Then, we illustrate the evaluation and enforcement of data freshness.

4.1. Determining the Appropriate Properties

Consider the following scenario for the evaluation of timeliness in the DIS of Example 1: By the characteristics of the sources (meteorological satellite and dissemination sources) data changes very frequently. In addition, the implementation of the system combines virtual and materialization techniques and synchronous and asynchronous pull-pull policies.

Lets consider that queries Q_1, Q_2, Q_3 and Q_4 have freshness requirements of 168, 72, 48 and 2 hours respectively; then, only the costs and delays of the same magnitude order are relevant. In this scenario, for calculating the *processing cost* of the activities, we consider a global cost that adds communication, computing and materialization costs. For calculating the *synchronization delay* we consider access constraints, materialization, synchronization policies and execution frequencies. We briefly describe such properties and their influence in the synchronization delay:

– *Synchronization policy*: It describes the synchronization between a node and a successor node. We distinguish four synchronization policies: (i) *synchronous-pull*, the successor asks the node for data, the latter executes and answers with the produced data, (ii) *asynchronous-pull*, the successor asks the node for data and the latter answers with its materialized data, (iii) *synchronous-push*: the node executes and sends the produced data to the successor, and (iv) *asynchronous-push*: the node sends its materialized data to the successor.

– *Execution frequency*: When an activity is not synchronized with predecessor nor successor nodes, we consider that it executes periodically, with a defined execution frequency.

– *Materialization*: It indicates if an activity materializes data or not.

– *Access constraints*: It is the lowest time interval that a source allows between two consecutive data extractions.

Example 2. Figure 4 shows the calculation dag of Figure 3 labeled with the *ExpectedFreshness*, *ActualFreshness*, *Cost* (processing cost), *Materialization*, *Efrequency* (execution frequency), *AccessConstraint*, *Spolicy* (synchronization policy), and *Delay* (synchronization delay) properties. The Spolicy property has the values Spull (synchronous pull) and Apull (asynchronous pull). The Materialization property has the values V (virtual) and M (materialized). The other property values are expressed in hours. □

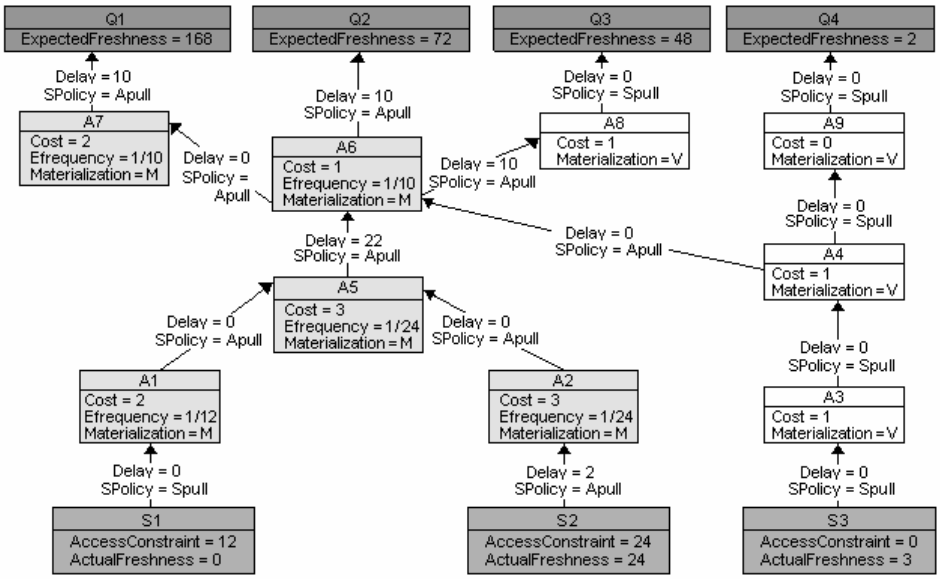


Figure 4 – Labeled calculation dag

In the presence of *access constraints*, the wrapper should periodically *materialize* data to assure the availability of source data. For example, consider that S_2 materializes data once a day and consequently it has no sense to query it more frequently (constraint of 24 hours), S_3 has no access constraint and can be queried at every moment (constraint is 0) and the system administrator has contracted to query S_1 every 12 hours because of its price.

When there are asynchronous synchronization policies between two consecutive activities, the data produced by the former must be materialized for being queried later by the latter, and could introduce synchronization delays. For example, the data produced by A_6 (see Figure 4) can have been materialized for almost 10 hours when read by A_8, then the delay will be 10 hours in the worst case. However, when there is no materialization, the activities execute immediately after its predecessors so there are no delays. Furthermore, activities having the same execution frequencies

(as A_6 and A_7) can be synchronized to execute one after the other without delay. When there is materialization, the delays with target nodes are the refreshment periods (inverse of refreshment frequencies) of the materialized data. When a source materializes data (as S_2) and it is read later by the wrapper there is a delay.

4.2. Evaluating Freshness

In this section we specify how to propagate freshness values within the graph, calculating the freshness of the intermediate data produced by each node. We firstly give an intuitive idea of the freshness calculation strategy and then we present an algorithm.

Intuitively, the freshness of the data produced by a node depends on the freshness of data at the moment of reading it (the freshness of data produced by the predecessor node plus the synchronization delay) and the time the node needs for executing (the processing cost). To calculate the freshness of a node we add such values. When the node reads data from several input nodes, input freshness values should be combined. As we are interested in an upper bound of freshness we take the worst case (the maximum).

The following pseudo-code sketches the algorithm:

```
FUNCTION DataFreshnessEvaluation (G: LabeledCalculationDag)
   RETURNS LabeledCalculationDag
BEGIN
   INTEGER value, max;
   FOR EACH source node A DO
      value= A.GetPropertyValue("ActualFreshness");
      A.addProperty("freshness", value);
   ENDFOR
   FOR EACH activity and target node A in a topological order DO
      max= 0;
      FOR EACH node B in G.getPredecessors(A) DO
         value= B.getPropertyValue("freshness")
                  + G.getEdge(B,A).getPropertyValue("delay");
         IF (value > max) THEN max= value;
      ENDFOR
      value= max + A.getPropertyValue("cost");
      A.addProperty ("freshness", value);
   ENDFOR
END
```

The *freshness* of a source node is its actual freshness. The *freshness* of an activity or target node is the maximum sum of the freshness of a predecessor node, plus the synchronization delay between nodes, plus the processing cost of the node.

Example 3. Figure 5 shows the Freshness property, calculated using the previous definition. For example, for A_6 both inputs are compared, 55 (32+22+1) and 6 (5+0+1) respectively, taking the maximum. □

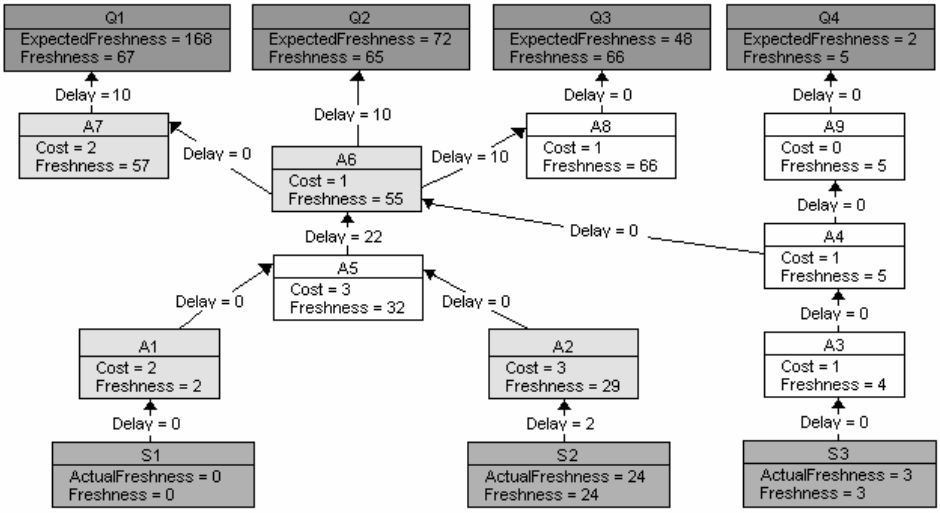


Figure 5 – Calculating freshness (Some properties have been omitted for readability)

Observe that for each node, there is a path for which we add all synchronization delays and processing costs to the source actual freshness and we obtain the freshness of the node. In the previous example, the freshness of A_7 can be calculated adding the freshness of source S_2 (24), plus the delays (2,0,22,0) and the costs (3,3,1,2) in the path [S_2, A_2, A_5, A_6, A_7]. In (Peralta et al., 2004) we show that there exists a path in the dag that determines the freshness of each node, and that this path is the most expensive one, i.e. the *critical path* (taking as costs the processing costs and synchronization delays). The existence of a critical path allows the use of a large spectrum of algorithms for optimizing a workflow of activities. Next section discusses their use when user requirements are not achieved.

4.3. Enforcing Freshness

The system should provide at the query level the data freshness expected by the users. To know if user freshness expectations can be achieved by the system, we can

propagate freshness values (executing the algorithm) and compare them with those expected for queries. If the propagated freshness values are lower than user expected values then freshness can be guaranteed. If the propagated freshness values are greater than user expected values, there exists at least one path from a source where propagated freshness is higher than expected freshness. There are several alternatives to enforce freshness in a path: negotiating with users to relax freshness expectations, negotiating with source data providers to relax source constraints, improving the design of the activities in order to reduce their processing cost and synchronizing the activities in order to reduce the delay between them (Peralta et al., 2004).

Example 4. In the example of Figure 5, data for solving Q_1 has a freshness of 69 hours which satisfy user expectations of 168 hours. This means that the system can be relaxed and activities in the paths from sources to Q_1 can be executed less frequently. Data for solving Q_3 does not meet user expectations (66 versus 48 hours). Analyzing the critical path to Q_3 ([S_2, A_2, A_5, A_6, A_8, Q_3]) some activities can be synchronized to reduce the delays and meet freshness expectations (for example executing A_6 immediately after A_5). The processing cost of some activities can be reduced too (for example replacing wrapper A_2 by a more performing one). However, even neglecting the cost of the activities in the paths to Q_4, freshness expectations cannot be achieved because the actual freshness of S_3 is too high. The solution should be a negotiation with users and/or source providers. □

5. Conclusion

Data freshness represents a family of quality factors and metrics. In this paper we have analyzed these factors and metrics and the features that influence the data freshness evaluation, namely system features, synchronization policies and nature of data.

We presented our current investigations in the line of developing an auditing tool for data freshness evaluation. We proposed a framework for performing such evaluation, which models the DIS integration process and its properties in terms of a labeled calculation dag. We discussed data freshness evaluation and enforcement solutions as graph traversal mechanisms and we illustrated our approach for a specific scenario.

Most of the functions of the previously described framework have been implemented in a Data Quality Evaluation tool (Fajardo et al., 2004). The prototype was implemented in Java (JDK 1.4); figures 4 and 5 are taken from the screens of the tool. The tool has been used to perform some experiences with data freshness metrics in different scenarios. The first experience was the evaluation of timeliness in a virtual scenario, neglecting processing costs and synchronization delays, i.e. propagating timeliness as the combination of source actual timeliness, as in

(Naumann et al., 1999). We have also used the tool with materialized scenarios, initially with asynchronous execution policies and progressively generalizing the scenarios to represent hybrid environments. The framework helped us to test the performance and appropriateness of the different evaluation algorithms. In the future, our goal is to confront the results with user quality profiles.

6. References

Ballow D., Wang R., Pazer H., Tayi, G., "Modelling Information Manufacturing Systems to Determine Information Product Quality", *Management Science*, vol. 44 (4), April 1998.

Baralis E., Paraboschi S., Teniente E., "Materialized view selection in a multidimensional database", in proc. of the *23rd Int. Conf. on Very Large Data Bases VLDB'97*, Greece, 1997.

Bouzeghoub M., Peralta V., "A Framework for Analysis of Data Freshness", in proc. of the *Int. Workshop on Information Quality in Information Systems IQIS'2004*, France, 2004.

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*, China, 2002.

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*, USA, 2000.

Fajardo F., Crispino I., Peralta V., "DWE: Una Herramienta para Evaluar la Calidad de los Datos en un Sistema de Integración", in proc. of the *X Congreso Argentino de Computación CACIC'04*, Argentine, 2004.

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*, USA, 1999.

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.

Gupta A., Mumick I., "Maintenance of Materialized Views: Problems, Techniques, and Applications", *Data Engineering Bulletin*, June 1995.

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*, Germany, 2003.

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.

Mannino M., Walter Z. (2004), "A Framework for Data Warehouse Refresh Policies", Technical report CSIS-2004-001, University of Colorado at Denver, 2004.

Naumann F., Leser U., "Quality-driven Integration of Heterogeneous Information Systems", in proc. of the *25th Int. Conf. on Very Large Databases VLDB'99*, Scotland, 1999.

Peralta V., Bouzeghoub M., Evaluating Data Freshness in Data Integration Systems, technical report, Université de Versailles, France, 2004.

Segev A., Weiping F., "Currency-Based Updates to Distributed Materialized Views", in proc. of the *6th Int. Conf. on Data Engineering ICDE'90*, USA, 1990.

Shin B., "An exploratory Investigation of System Success Factors in Data Warehousing", *Journal of the Association for Information Systems*, vol. 4(2003), 141-170, 2003.

Theodoratos D., Sellis T., "Data Warehouse Configuration", in proc. of the *23rd Int. Conf. on Very Large DataBases VLDB'1997*, Greece, 1997.

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*, Germany, 1999.

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.

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*, UK,1997.