IQIS 2005 Keynote Speech 1
Handling Data Quality in Entity Resolution

Hector Garcia-Molina

Stanford University
California, USA

ABSTRACT

Entity resolution (ER) is a problem that arises in many information integration scenarios: We have two or more sources containing records on the same set of real-world entities (e.g., customers).

However, there are no unique identifiers that tell us what records from one source correspond to those in the other sources.

Furthermore, the records representing the same entity may have differing information, e.g., one record may have the address misspelled, another record may be missing some fields.

An ER algorithm attempts to identify the matching records from multiple sources (i.e., those corresponding to the same real-world entity), and merges the matching records as best it can.

In many ER applications the input data has data quality or uncertainty values associated with it. Furthermore, the ER process itself introduces additional uncertainties, e.g., we may only be 90% confident that two given records actually correspond to the same real-world entity.

In this talk Hector Garcia-Molina will discuss the challenges in representing quality/uncertainty/confidences in a way that is useful for the ER process.

He will also present some preliminary ideas on how to perform ER with uncertain data. (This work is joint with Omar Benjelloun, David Menestrina, Qi Su, and Jennifer Widom).

IQIS '05, June 17, 2005, Baltimore, MD, USA.

ABOUT HECTOR GARCIA-MOLINA

Hector Garcia-Molina is the Leonard Bosack and Sandra Lerner Professor in the Departments of Computer Science and Electrical Engineering at Stanford University, Stanford, California. He was the chairman of the Computer Science Department from January 2001 to December 2004. From 1997 to 2001 he was a member the President's Information Technology Advisory Committee (PITAC).

From August 1994 to December 1997 he was the Director of the Computer Systems Laboratory at Stanford. From 1979 to 1991 he was on the faculty of the Computer Science Department at Princeton University, Princeton, New Jersey. His research interests include distributed computing systems, digital libraries and database systems.

He received a BS in electrical engineering from the Instituto Tecnologico de Monterrey, Mexico, in 1974. From Stanford University, Stanford, California, he received in 1975 a MS in electrical engineering and a PhD in computer science in 1979.

Garcia-Molina is a Fellow of the Association for Computing Machinery and of the American Academy of Arts and Sciences; is a member of the National Academy of Engineering; received the 1999 ACM SIGMOD Innovations Award; is a member of the Computer Science and Telecommunications Board (National Research Council); is on the Technical Advisory Board of DoCoMo Labs USA, Kintera, Metreo Markets, TimesTen, Verity, Yahoo Search Marketplace; is a Venture Advisor for Diamondhead Ventures, and is a member of the Board of Directors of Oracle and Kintera.

IQIS 2005 Keynote Speech 2
Methods and Analyses for Determining Quality

William E. Winkler

U.S. Bureau of the Census,
Statistical Research
USA

ABSTRACT

In a possibly ideal world, records in a database would be complete and would contain fields having values that correspond to an underlying reality. An individuals name, address and date-of-birth would be present without typographical error. An income field might be a reasonably close approximation of a "true income" and would not be missing. A list of customers would be complete, unduplicated and current.

In this ideal world, a database could be used for several purposes and would be considered to have high quality. A set of databases might be linked using name, address, and other weakly identifying information.

In this paper, we describe situations where properly chosen metrics may indicate that data quality is not sufficiently high for monitoring processes, for modeling, and for data mining.

Some of the metrics are supplementary to those in the quality literature or have rarely been used. Additionally, we describe generalized methods and software tools that allow a skilled individual to perform massive clean-up of files in some situations.

The clean-up, while possibly sub-optimal in recreating "truth", can replace exceptionally large amounts of clerical review and allow many uses of the "cleaned" files.

IQIS '05, June 17, 2005, Baltimore, MD, USA.

ABOUT WILLIAM E. WINKLER

Ph.D. Probability Theory
Principal Researcher, US Census Bureau
Fellow, American Statistical Association

Provider issues in quality-constrained data provisioning

Paolo Missier and Suzanne Embury
School of Computer Science
The University of Manchester, UK
{s.embury,pmissier}@cs.manchester.ac.uk

ABSTRACT

Formal frameworks exist that allow service providers and users to negotiate the quality of a service. While these agreements usually include non-functional service properties, the quality of the information offered by a provider is neglected. Yet, in important application scenarios, notably in those based on the Service-Oriented computing paradigm, the outcome of complex workflows is directly affected by the quality of the data involved. In this paper, we propose a model for formal data quality agreements between data providers and data consumers, and analyze its feasibility by showing how a provider may take data quality constraints into account as part of its data provisioning process. Our analysis of the technical issues involved suggests that this is a complex problem in general, although satisfactory algorithmic and architectural solutions can be found under certain assumptions. To support this claim, we describe an algorithm for dealing with constraints on the completeness of a query result with respect to a reference data source, and outline an initial provider architecture for managing more general data quality constraints.

1. INTRODUCTION

An increasing number of information providers nowadays offer query services on large data sets through internet-wide published interfaces, using a variety of widely available technologies. Alongside the definition of a service interface, the stipulation of agreements regarding the quality of the service is also becoming commonplace, eg. in the form of *Service Level Agreements* [12, 2, 20]. Such agreements, however, only deal with performance issues, while the quality of the information delivered to service users is generally neglected. When compared to the more common experience of shopping for any kind of product, this situation is akin to assuming that the customers' only issue is with the opening hours of the store or the service time at checkout, with no concern for the quality of the goods – clearly an unrealistic expectation.

IQIS 2005, June 17, 2005, Baltimore, MD, USA.

We argue that data consumers are in a similar predicament: the sizable and mature body of knowledge regarding quality properties of data [18, 19, 24, 21, 9] does not translate into actionable user requirements, and yet, in simple and realistic scenarios, specific properties of data are important. Suppose, for instance, that a provider acquires copyrighted articles from publishers, and compiles independent digests and reviews of those articles, offering them for sale. While users are interested in purchasing the added-value reviews, they also want to make sure that by doing so, they are not missing the digest for any of the articles that would meet their criteria if requested directly to the publishers. For example, they want to purchase the digest for the ten most recent papers on a particular topic.

The idea at the core of our work is that users may enforce this and similar requirements by entering into a formal agreement with the added-value provider, whereby the digests produced in response to a query are guaranteed to include a sufficiently large fraction of the articles that would have been returned, had the same query been issued directly to the publisher. We refer to this property of the data as its *completeness* relative to a reference data source –in this case, the original publisher.

Thus, a completeness constraint is intended to differentiate between providers that only offer digests for a small subset of the articles that are actually available, and those that account for larger collections. Notice that, in this example, the quality of the digest itself is not part of the agreement, although it may be similarly formalized as a quality constraint, of a different type: completeness is only one of many possible properties of data for which constraints can be defined.

This simple scenario is becoming increasingly relevant in situations where (i) the data obtained from a provider has a quantifiable value to the consumer, (ii) its worthiness depends on one or more quality properties, and (ii) multiple providers may offer similar information. The combination of these factors contributes to the development of a data marketplace, whereby consumers that are interested in quality data negotiate its quality/cost trade-offs with providers. The value of quality as perceived by consumers is not necessarily only monetary. Consider for instance the case, also increasingly important, of a biologist who performs data-intensive experiments using various algorithms that operate on data obtained from public repositories (so-called *in silico* experiments). For example, the success of a gene sequence similarity analysis, consisting of matching a string sequence against a large database of known sequences, depends on

the completeness of the reference data source. Although the experiment's success criteria are normally not expressed in monetary terms, the value of using a complete reference data set is unquestionable.

The missing element that would enable data marketplaces is the ability for data providers and consumers to negotiate formal and binding agreements regarding the quality of the data. In this respect, providers seem to face the greatest challenges, as they must determine which agreement levels can be sustained, and the cost/benefit trade-offs involved. To the best of our knowledge, these issues have not been addressed, with the exception of a 1989 paper by Ballou and Tayi [5], discussed later.

This paper attempts to fill this gap. Its core contribution is a model for quality agreements, and an analysis of the issues and possible strategies available to providers that commit to such agreements. We begin by assuming that every data transaction, consisting of a query-result pair, may be subject to quality constraints. Before any such transaction may occur, providers and consumers should agree on definitions for the following elements:

- a pricing function, which associates a value to the result of any query issued by the consumer;
- a number of quality functions that formalize the notions of quality properties, and that associate a quality value to each result;
- a function of quality values for the result, that quantifies their appropriateness to the consumer. This function, not necessarily linear, is expressed as a *penalty* whose effect is to reduce the price paid for the result.

The negotiation process that leads to the specific definition of each of these elements is not relevant for our purposes, and is not considered in this paper. Before entering into such an agreement, the provider must determine its feasibility, by assessing (i) the actions required to provide data with the required quality values, and (ii) whether its data architecture supports those actions in a cost-effective way. Specifically, for each incoming query, the provider faces two problems:

1. **Detection:** it must determine to what extent a result set would incur any penalty, due to insufficient quality levels, for all the quality properties involved;
2. **Repair:** it must determine what actions are available to repair its data in order to reduce or avoid the penalties.

We base our model on the assumption that both detection and repair have a cost, forcing the provider to solve an optimisation problem involving penalties, price, and costs.

As is common with any marketing scenarios, the provider may adopt a number of different strategies for compliance. For instance, it may invest resources to proactively ensure that most of its data comply with the constraints, regardless of the specific user requests. More realistically, it may conservatively adjust the quality levels of its data, by observing the consumer's behaviour – for instance, by investing in quality only for the most popular data and accepting penalties for less frequently requested items.

Rather than focusing on any specific model, in this paper we define the provider's problem space by enumerating the factors that affect its strategies. This results in a general framework that can be used to analyze complex scenarios. The most critical factor concerns the amount of information available to ensure that the penalties assessed are *provably fair*: if we assume that the actual value of a quality function is always available both to the provider and the consumer, then (i) the fairness of the agreement can be verified, and (ii) the provider may implement an optimal provisioning strategy. For some quality properties, however, this assumption may not be realistic. For instance, evaluating the completeness of a data set relative to a reference set requires full knowledge of the latter. We model the problem of partial knowledge of quality by attaching a cost to the evaluation of quality functions; in the case of completeness, this would be the per-item cost of querying the reference source, in order to obtain a partial view of its contents. This leads to the formulation of quality estimates, which put the fairness of the agreement into question, and compromise the optimality of the provider's strategy. We propose (Section 3.1) to deal with fairness issues by introducing a third-party verification role and using a spot-check approach for ensuring limited but acceptable fairness.

As additional contribution, we present an algorithm for dealing with the specific case of data completeness constraints, first using the most favorable assumptions, including availability of quality values, and then in the more general case of partial availability. This provides an insight into the expected complexity of the general provisioning problem, and also shows a case of a property-specific algorithm that cannot be easily reused for other properties; it suggests that the generality of the solution may be limited to the detection-maintenance pattern.

As a final contribution, we describe (Section 6) a general data provider architecture that incorporates that pattern, and show how it can be implemented alongside a standard query processing engine.

Our initial investigation into the problem of provisioning data with quality constraints shows that this is a difficult one in general, although satisfactory algorithmic solutions can be found under realistic assumptions.
In the rest of the paper, quality functions are introduced in Section 2, followed by the agreement model in Section 3. The provider model is presented in Section 4, and the specific handling for completeness in Section 5. The reference architecture is discussed in Section 6. We conclude in Section 7 with our agenda for further work.

1.1 Related work

Although the specific topic of data quality agreements is new, some authors address related problems, specifically in the area of quality-aware data integration. Naumann et al. [17] assume that scores can be assigned to the data offered by multiple providers, to reflect its quality, and show that the problem of data integration in the presence of such scores results in significant extensions to known algorithms for querying data using views. We tackle a somewhat complementary problem, namely how a provider can manage its data assets in order to *obtain desirable quality scores*, which would then be passed up to an integration mediator. Similarly, a recent paper by Motro [3] shows how using utility functions may alleviate the problem of data fusion (i.e., combining different versions of the same date) in the presence of inconsistencies. Utility functions are based on quality features such

as recentness and accuracy. Using a similar perspective, the "Quality Broker" architecture presented in [15] assumes that quality features are available from several sources. The goal of the architecture, in this case, is not quality constraint satisfaction, but rather the broker-based selection of the most appropriate answer to a query, among multiple available.

A related problem is also addressed in [4], where the notion of a *data quality certificate* is presented. The purpose of the certificate is to enable reasoning about quality within the context of cooperative information systems, in order to improve the overall quality of inter-system workflows. This notion is also used, in a different form, in the quality profile model described in [13].

Underlying all of these approaches are assumptions regarding (i) a shared underlying model for quality description, and (ii) the way quality values are actually computed. While none of them seems concerned with their actual availability, in some cases, the granularity of the quality meta data is so fine – at the level of a single attribute, that the actual feasibility of computing the corresponding values may be questioned. An important but overlooked problem then becomes, to assess the robustness of these integration processes when some of the quality data is missing.

Some authors define completeness by taking into account both the size of a data source and the number of available attributes with respect to the reference source, as well as the fraction of attribute values that are non-null. Our definition of completeness only considers the size of the relation, and not the single attributes, and thus it is simpler than the *relational completeness* found in [16]. It corresponds roughly to the notion of *coverage* introduced in [11]; there, coverage is defined with respect to a universal relation, whose extension includes all tuples obtained from a number of primary data sources. Our single reference source corresponds to the universal relation.

Ballou et al. [5, 6] presented an interesting and very pertinent early attempt at linking quality properties to the effort required to provision them. There, the goal is to determine, using an integer programming model, the most effective distribution of resources that a provider can use to maintain or enhance data integrity, each with an associated cost and effectiveness. While initially assuming precise knowledge of the underlying cost, data error rate and other parameters, the model also addresses the problem of estimating some of those parameters using heuristics. How realistic the overall model is in practice, however, remains to be seen.

Finally, following the intuition that information is but another type of product, some authors have adapted established results from the practice of quality control in product manufacturing, resulting in the IP-MAP (Information Product Map) framework [23, 22, 7]. For our purposes, the merit of this work is to provide ways to *predict* quality values based on the analysis of the processes that produce those values. However, we believe that the barebone model for completeness presented in the next section would defeat the purpose of such machinery, which is best suited for complex processes with well-identified "quality weak spots."

2. QUALITY FUNCTIONS

We begin by providing functional definitions of quality properties, which we illustrate for the case of completeness, defined earlier, and of consistency, i.e., the property of a data set to conform to some validation rule.

We only consider relational data sets, i.e., extensions of a relational schema. Given two data sets D and D_r, we define the completeness of D relative to D_r as:

$$compl(D, D_r) = \frac{|D \cap D_r|}{|D_r|} \in [0, 1] \quad (1)$$

In particular, we are interested in the completeness of the result $Q(D)$ of a query:

$$compl_Q(D, D_r) = \frac{|Q(D) \cap Q(D_r)|}{|Q(D_r)|} \in [0, 1] \quad (2)$$

Intuitively, $compl_Q()$ counts the fraction of records from D_r that the user obtains by querying D rather than D_r. Recall that the reason for querying D in our examples is that it contains *added value* versions of data from D_r.

This definition is illustrated in Figure 1. Note that $compl(D, D_r)$ may be very different from $compl_Q(D, D_r)$ for some Q; even when D contains only a small subset of D_r, its completeness relative to a particular query may be high, as long as D contains most of the items that the user is requesting. In fact, the heuristics for providing completeness, presented later, are based on the provider's knowledge of the user queries; their effectiveness depends on the predictability of those queries.

Before we proceed, we must first give a precise meaning to the expression $Q(D_r)$, by clarifying the relationship between D and D_r. Let S_D and S_{D_r} be the relational schemas for D and D_r, respectively. Following the well-known "Global-as-View" pattern, described for instance by Lenzerini [14], we assume that S_D is defined as a view on S_{D_r}. Formally, this requires the definition of a mapping query mq over S_{D_r}, written as $S_D \rightsquigarrow mq(S_{D_r})$, so that any query issued to S_D can be translated into a corresponding query to S_{D_r}, through a simple process of *unfolding* of the mapping query. For example, suppose that S_{D_r} consists of two relations, $R_1(a_1, a_2, a_3)$ and $R_2(b_1, b_2, b_3)$, and that S_D consists of relation R, defined by the mapping query:

$$R(c_1, c_2, c_3) \rightsquigarrow \pi_{a_1,b_2,b_3}(\sigma_{a_3=c}(R_1 \bowtie_{a_2=b_2} R_2))$$

Then, for a query like

$$Q \equiv \pi_{c_1}(\sigma_{c_2=x}(R)),$$

the corresponding query on S_{D_r} used in the completeness definition would be

$$Q' \equiv \pi_{a_1}(\sigma_{b_2=x \wedge a_3=c}(R_1 \bowtie_{a_2=b_2} R_2))$$

To simplify the notation, in the rest of this paper we write $Q(D_r)$ instead of $Q'(D_r)$.
It is also worth mentioning that $Q(D) \cap Q(D_r) = Q(D) \cap D_r$: the use of $Q(D_r)$ only really matters in the denominator of expression (2).

As a second example of functional definition of quality property, we also define the consistency of a data item relative to a conformance rule; for instance, a rule may state that a street address in a database entry is consistent with a reference street atlas, if it can be matched uniquely against one of the reference streets in the atlas. The record matching problem has been studied extensively in the data quality literature [10, 8, 1] [1], and it is known that the evaluation of

[1]W.Winkler has made a rich collection of references to this problem available at http://csaa.byu.edu/kdd03-papers/winkler-refs.pdf.

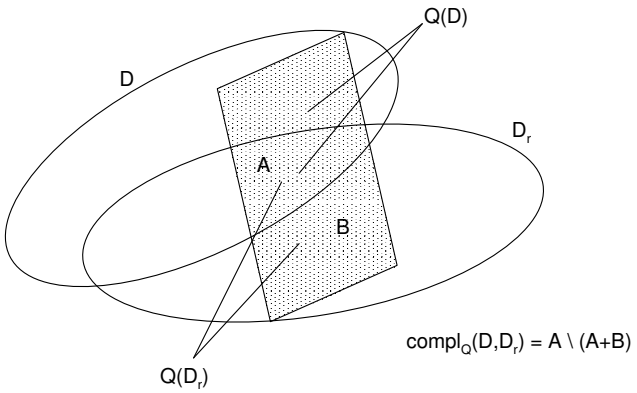

Figure 1: Completeness of a query result

this rule may incur uncertainty, accounting for the chance of false positives (an erroneus match). Thus, we assume that the rule is a function of the data and of some parameters (i.e., the reference source), and that its evaluation produces a "yes/no" result, along with a level of confidence. This function does not perform any correction or issue recommendations. In this regard, it behaves like an integrity constraint checker on a database schema.

The validation function for an item $d \in D$ with respect to D_r is

$$val(d, D_r) \in (\{true, false\}, [0, 1])$$

where the second component of the value is the confidence in the outcome. Given a user-defined threshold c_0 for the confidence, let the set of *acceptable* items relative to *val* and c_0 be

$$acc(val, D, D_r, c_0) = \{d \in D | val(d, D_r) = (true, c) \wedge c \geq c_0)\}$$

A simple definition for the consistency of D relative to D_r, given c_0, is the proportion of acceptable items in D. For $Q(D)$ this is written as:

$$cons_Q(val, D, D_r, c_0) = \frac{|acc(val, Q(D), D_r, c_0)|}{|Q(D)|} \in [0, 1]$$

This simple definition illustrates the general idea of consistently defining normalized quality functions, using user-specified parameters such as the threshold c_0.

3. AGREEMENT MODEL

Agreements are based on a simple penalty/reward model, whereby the consumer and the provider agree on formally defined quality constraints for the data provisioned on a query-by-query basis, the provider may charge fees in return for data, and the consumer may assess penalties when the quality constraints are violated.

Rather than defining discrete constraints, i.e., of the form $compl_Q(D, D_r) > compl_{min}$, we instead allow for a more general formulation, by associating penalty functions of arbitrary shapes to quality functions. When applied to a base price for a query result, they reduce the actual fee paid, in a way that is proportional to the perceived importance of the specific property.

For query Q on D, the agreement includes the following elements:

- a set of normalized quality functions of the form

  $$qf(D', \mathcal{P}) \in [0, 1]$$

  for any $D' \subseteq D$, where 1 is the best quality achievable. $\mathcal{P}$ indicates a property-specific set of additional parameters, eg. D_r, c_0. In particular, we are interested in computing the quality associated to a query result, $qf(Q(D), \mathcal{P})$;

- a base pricing function $price_b(D')$ for any $D' \subseteq D$. Again, in practice we are interested in computing $price_b(Q(D))$;

- a penalty function $pen_{qf}(D', \mathcal{P}) \in [0, 1]$, which introduces a bias on the base price, by mapping the values of the normalized quality function qf applied to D', onto a penalty factor.

The actual price paid by the user for $Q(D)$ is then

$$price(Q(D), qf(), \mathcal{P}) = price_b(Q(D)) \times (1 - pen_{qf}(Q(D), \mathcal{P}))).$$

Note that discrete quality constraints can be expressed simply by defining binary penalty functions. For example, the following constraint makes the result set worthless if the completeness falls below a threshold $compl_{min}$:

$$pen_{compl}(D', D_r) = \begin{cases} 0 & \text{if } compl(D', D_r) > compl_{min} \\ 1 & \text{otherwise} \end{cases}$$

Normally, however, the penalty will be proportional to the quality level, i.e., the function is monotone decreasing in the value of $qf()$. Although no assumptions on the shape of penalty and pricing functions need to be made, this additional information helps in reducing the complexity of the provider algorithms for managing quality compliance. The baseline algorithm presented in Section 5.1, shows a worst case scenario, in which no assumption is made regarding the shapes of these functions.

Note also, that using normalized quality functions makes it straightforward to extend this pricing scheme to multiple quality properties, i.e., by combining multiple quality and penalty functions:

$$price(Q(D), \{qf_i()\}, \{\mathcal{P}_i\}) = price_b(Q(D)) \cdot \prod_i (1 - pen_{qf_i}(Q(D), \mathcal{P}_i)))$$

Functions $\{qf()_i\}$, $price_b()$ and $\{pen_{qf_i}()\}$ are all defined as part of a negotiation process, whose details are not relevant for our purposes. Once the agreement is in place, it is enforced as follows:

- for every incoming query Q, the provider computes $D' = Q(D)$;
- for every $qf_i()$ that is subject to a penalty, compute $q_i = qf_i(D', \mathcal{P}_i)$ and $p_i = pen_{qf_i}(q_i)$;
- compute the final price $price_b(D') \cdot \prod_i (1 - p_i)$.

3.1 Fairness of penalty assessment

As anticipated, the fairness of the penalty assessment depends upon the value of $qf()$ being available. In our example, this corresponds to having a catalog of all available articles, which can be queried (this is $Q(D_r)$), regardless of how many of those articles have received a review. Similarly, in the biology database case, a public source for the primary data may be available free of charge. While these are reasonable assumptions, in general we also need to consider the cost of computing $qf()$. When the provider incurs this *monitoring cost* (see Section 4), it may need to compute an estimator $\hat{qf}()$ of the actual $qf()$, balancing its precision with the cost of limited monitoring.
The idea is then to let providers self-assess their penalties, and to adopt the simple but widespread view that a third-party verification authority is in charge of assessing the correctness of penalties. We assume that this authority also incurs a monitoring cost. For completeness, this results in the following scenario:

- the provider defines its own estimators $\hat{qf}()$ for $qf()$, based on some probabilistic model and by sampling using a limited number of $Q(D_r)$ queries, determined by its budget[2];
- the authority has its own strategy for verification, which relies on spot-checks performed at some time intervals, again by querying $Q(D_r)$;
- the provider may negotiate a tolerance $\delta \in [0, 1]$ as part of the agreement, which limits its liability vs the customer in case of imprecise estimates;
- whenever the authority determines the actual value for $qf()$, the percentage estimation error $\hat{e} = \frac{\hat{qf}() - qf()}{qf()}$ is computed, and the provider incurs a fine that is proportional to $\hat{e} - \delta$, whenever $\hat{e} > \delta$.

As a result, the provider's chance of getting away with an incorrect estimate (and hence, a reduced penalty) depends on the authority's budget and ability to monitor effectively.

In this scheme, the consumer relies on the authority for control. In case of dispute of past transactions, the authority is obliged to perform a check on a past quality value, which may require enabling infrastructure. For completeness, this amounts to querying a past state of the D_r database – which is feasible if D_r is a standard transactional DBMS with logging capabilities.

[2]We are implicitly assuming, for the purpose of the example, that the monitoring cost in this case is proportional to the number of items retrieved from D_r.

4. PROVIDER COMPLIANCE MODEL

In this section we analyze the issues associated to enforcing an agreement, from the provider's perspective. The quality-constrained data provisioning problem can be described according to a simple "monitor-assess-repair" reactive model:

monitor: Firstly, the provider must compute the value of quality functions every time it receives a query. Since some of the function parameters may not be available, i.e., $Q(D_r)$, they must be estimated;

assess: Secondly, the provider must estimate the penalty associated with the result set for the query;

repair: Thirdly, it must determine the most cost-effective repair actions to be executed in order to move the state of its data set towards compliance.

With reference to completeness and consistency, the model is instantiated as follows.

Detection. We denote with $\overline{D}_Q$ the quantity we wish to monitor. For completeness, this quantity is

$$\overline{D}_Q, compl = Q(D_r) \setminus Q(D)$$

This set contains all the items that the user would have obtained by issuing Q to D_r, but are instead missing from $Q(D)$. These are therefore the items responsible for the penalties incurred when returning $Q(D)$. For consistency, $\overline{D}_Q$ is the set of items that were expected to be consistent, but are not:

$$\overline{D}_Q, cons = Q(D) \setminus acc(val, Q(D), D_r, c_0)$$

Repair. For completeness, the only repair procedure consists in obtaining new data items from D_r. For consistency, the procedure may perform validation on items whose consistency is unknown, and apply algorithms to enforce consistency (for instance, by correcting data or obtaining new versions from a third party source).

Costing. The third step is the choice of a provider cost model, which includes a *monitoring* component $cost_m(Q, D)$, a *repair* component $cost_r(D)$, and the *value-adding* component $cost_{va}(d)$ of expending local resources in order to prepare any $d \in D$ for delivery.[3] For completeness, the first two correspond to the cost of computing $\overline{D}_Q, compl$ and the cost of obtaining a new item d from D_r, respectively.

Compliance strategies. Next, the provider must identify a strategy for activating repair procedures given the price and penalty information from the agreement, the observed violations, the cost model, and a goal. An obvious general provider goal is to avoid penalties that erode profit, by incurring the minimal repair cost. For completeness, this translates into the strategy of obtaining the smallest set of items from D_r, which restores the required completeness levels. As noted, D may be a small subset of D_r, however if it contains the items that are most likely to be requested in the future, these may be sufficient to satisfy the constraints for most of the user queries. Therefore, a sensible approach is to use the history of past queries to estimate the likelihood of any item in D_r being requested in the future. Estimating future request probability is clearly more effective than a simpler greedy strategy, which would acquire only the items

[3]This could for example be the cost of producing a review for a new article.

that are missing from the current query, some of which will not be requested again.
Regardless on the specific choice of estimator, the precision (i.e., confidence level) associated to the estimate depends on the length of query history: for a new agreement or a new user, the provider will be able to do little more than repairing based on the current query. Various statistical models can be developed to estimate such probability, and it is not the purpose of this work to survey them. Simple estimators include the frequency of past requests, which assume that the past popularity of an item is an indicator of future interest; a mobile average on a limited time window, which attempts to track the changes in interest; or an estimator based on the hypothesis that the occurrence of a request has a known distribution. We note in passing the similarity between the problem of predicting the request of items that are obtained from reference sources, and the problem of defining cache replacement algorithms: for properties like completeness, similar estimators may be applicable.

4.1 Factors that affect compliance strategies

A number of factors complicate the choice and implementation of a strategy:

- **Instant repair:** is it feasible for the provider to obtain new items from D_r, and use them to repair the current result set? When this is not possible, current penalties are inevitable, and the repair strategy may only focus on avoiding future penalties.

- **Completeness of detection:** has the provider complete knowledge of the quality indicators? The provider must balance the precision of the $\overline{D}_Q$ estimate, with the monitoring cost $cost_m()$ of computing it, and the chance that the verification authority will claim irregularities.

- **Number of quality properties** that appear in a single agreement, or in multiple agreements: the potential interaction between constraints on different properties may complicate the strategy. Consider for instance *currency*, the property of a data value of being correct at a given time[4]. Currency and completeness are not independent, because if we assume that all items in D_r are current, then obtaining a new item from D_r in order to restore completeness, has also the effect of improving currency. On the other hand, given a budget for acquiring new items, there may be a contention between different quality constraints that depend on those items for their satisfaction, breaking the isolation of single-property strategies.

- **Options available for detection and repair:** while for completeness the only repair option is to obtain items from a reference source, multiple such sources may be available, possibly at different costs. Also, other properties may present richer options: validating an item for consistency may involve requesting a correction from a reference source, or performing manual inspection on the item.

[4]The data for an address that changed recently may have been correct before the change occurred, but it has since become obsolete, or non-current, until it is updated.

- **Notification of updates to a reference source:** for properties whose repair actions depend on a reference source, it matters whether the provider is notified of any update to the source. For instance, if completeness is estimated by periodically sampling D_r, then the estimate is affected by the frequency of updates to D_r.

- **Shape of cost and price functions:** the various cost and price functions listed earlier may depend on the particular choice of item, or may be defined as a function of the size of the result.

5. PROVISIONING WITH COMPLETENESS

In order to provide a concrete example of provisioning with quality, we now illustrate an algorithm for completeness, based on the detection-repair model. With respect to the complicating factors listed above, the assumptions for the algorithm are as follows: instant repair is possible; only one quality property appears (completeness), and the only repair option is to obtain new items from the reference source; there is no notification of updates to the reference; and finally, the price function depends only on the size of the query result.

We first present a baseline algorithm that assumes that the provider has complete and free knowledge of the quality indicators, and then propose a generalization that does not require this assumption.

5.1 Baseline algorithm

The approach is based on the definition of a utility function for a set $D' \subseteq D_r \setminus D$ of currently missing items, and the formulation of a corresponding optimization problem, that can be solved using heuristics. For completeness, the function takes into account the provider cost model and the penalty functions:

$$\begin{aligned} U(D', Q, D, \mathcal{P}) \;=\; & price_b(Q(D) \cup (D' \cap Q(D_r))) \\ & \times (1 - pen_{compl}(Q(D) \cup (D' \cap Q(D_r)), \mathcal{P})) \\ & - cost_r(D') - cost_{av}(D') \end{aligned}$$

In practice, U describes the effect of purchasing set D':

- the base price is increased due to the new items in D'. Notice that there is no guarantee that the algorithm will only purchase items that are missing from the current result. In fact, the heuristic presented later makes a less greedy selection, hoping to improve *future* compliance. Hence, only the items in $D' \cap Q(D_r)$ contribute to the immediate extra reward.

- The penalty is reduced correspondingly (the same observation applies).

- The cost incurred is due to purchasing and adding value to D'.

The optimization problem is designed to limit the risk of purchasing items that may not be needed in the future: we are seeking the subset of $D_r \setminus D$ that maximizes the ratio of utility-to-size:

$$\max_{D' \in D_r \setminus D} \frac{U(D', Q, D, \mathcal{P})}{|D'|}$$

Normalizing by size avoids the effect of an indefinitely increasing utility, which would result in purchasing the largest possible D'. Note that the problem is defined on the entire set of missing items, rather than only on $\overline{D}_Q$, and that it must be solved when Q is computed.

Since we are not making any assumptions on the shape of function U, or of any of its components, a brute-force algorithm that enumerates all possible D' has exponential complexity in $|D_r \setminus D|$. To reduce the complexity, we apply the strategy mentioned in Section 4, using the history of past user queries to estimate the likelihood of a missing item to be requested in the future.

Algorithm 1. For each Q, the provider maintains a count of the frequency of requests of each item $d_i \in \overline{D}_Q$, and requires an estimate of the likelihood of a future request for each d. Since this involves updating the frequency of the items that are actually requested, complete knowledge of D_r is not required. The algorithm starts from an empty set D', and incrementally adds to it in order of estimated likelihood, recording the value of U at each step. In this way, at step i only the most promising of the $\binom{|D_r \setminus D|}{i}$ potential sets is considered. □

Some comments are in order:

- From the definition of the utility function, we note that its components depend not only on the number of items, but also on their choice. This is because we allow the selection of D' to range on the entire set $D_r \setminus D$, rather than only on $\overline{D}_Q$, hence some of the selected items may not reduce the immediate penalty.
- The order defined on the missing items is partial. For instance, after the first query, the best set contains a random selection (of optimal size) of items from $\overline{D}_Q$, because all such items have the same frequency of occurrence. We assume that the items in $\overline{D}_Q$ are preferred over others of the same rank.
- It is worth considering the effect of a *locality principle* on this heuristic, which states that the history of past queries is indeed a good predictor for future queries. Consider what happens when queries are highly localized and an occasional odd query arrives, requesting items never mentioned before. Since these items have a low frequency, they are ranked low relative to others that have been requested in the past multiple times, but are still missing. In this case, the algorithm does not try to repair the current query (which will therefore result in a penalty), but rather it will purchase additional popular items, increasing the expected reward for future queries.
- As noted earlier, initially the limited history of past requests makes the estimators unreliable, yielding items that may in fact never be used again in the future. This confirms the intuition that this agreement model makes frequent and regular consumers more appealing than occasional ones, and suggests that quality agreements are best suited for long-term consumer-provider relationships.

We conclude by noting the effect of updates and inserts into D_r. In this version of the algorithm, even if the provider is not informed of these events (for instance, through some event notification infrastructure), they do not pose problems.

When an update comes to D_r, then D clearly holds a stale copy, of which it is not aware. However, unless there is an explicit currency constraint, this has no consequences on completeness! – this is in fact a case for handling completeness and currency together. When a new insert occurs in D_r, according to our algorithm it may be revealed only through queries of the form $Q(D_r)$. Items that appear in D_r but are never requested, are simply ignored. Items that start being requested after they have been inserted, are handled in the same way as all others.

5.2 Ranking of missing data using query predicates

Given a query of the form $Q = \sigma_p(R)$, our baseline algorithm relies on the *extension* D_r for computing completeness (detection), and for selecting the most promising items to purchase (repair), assuming $Q(D_r)$ known, by simply enumerating the missing items. The algorithm described in this section addresses the problem of performing detection and repair when $Q(D_r)$ is not available.

The idea is to consider only the *conditions* stated in the user queries, and those used by the provider to obtain new items from D_r. We rely on two observations: firstly, that the most popular data are represented at the *intensional* level by the history of user queries; and secondly, that although $Q(D_r)$ is not immediately available, within the limited scope of a specific user query we may still provide a good estimate for the completeness $compl_Q(D, D_r)$, and also determine the conditions corresponding to the most popular items in D_r, for repair.

The algorithm is based on the definitions of *request profiles* and *completeness maps*. Similar in spirit to ordinary database profiles used by relational query optimizers, a request profile records the level of interest for specific data items, and is computed progressively from a history of user queries. The main difference is that, while in ordinary profiles the data points in the histograms are attribute values, in this case they are *query predicates*.

Whereas a request profile records the demand for data, a completeness map represents the available data set, as described by the set of queries issued by the provider to D_r. Intuitively, knowledge of the user requests to the provider and of the provider requests to its suppliers is sufficient to rank the data that the provider is missing, according to the user preferences.

Let $\mathcal{Q} = Q_1, Q_2, \ldots, Q_m$ and $\mathcal{Q}' = Q'_1, Q'_2, \ldots, Q'_n$ be the history of user and provider queries, respectively. We may restrict our attention to select-queries only, of the form $Q = \sigma_p(R)$, ignoring projections; having defined completeness at the granularity of the entire data item, a distinction based on projected attributes is unnecessary. Furthermore, we make the simplifying assumption that selection predicates are conjunctions of elementary conditions that are either (i) expressions involving relational operators *relop* $(=, \leq, \geq)$ on ordered domains, of the form x *relop* c, or (ii) set membership expressions on enumerated sets, i.e., $x \in \{c_1, \ldots, c_n\}$.

Given a relation $R(A_1, \ldots, A_l)$, these definitions are formalized as follows.

DEFINITION 1. *(Request profile)*

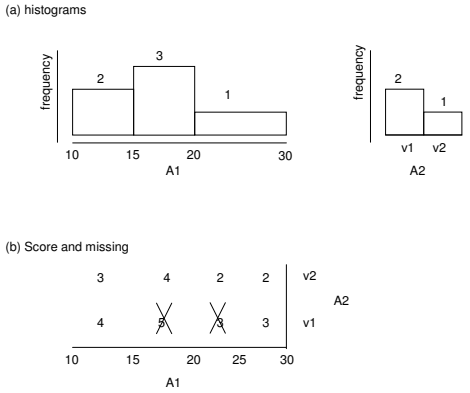

Figure 2: Construction of request profiles

Given set $P_i = \{p_{i1}, \ldots, p_{in_i}\}$ of logically disjoint predicate expressions p_{ij} on A_i, a request profile is a set of l histograms

$$freq_i : P_i \rightarrow \mathcal{N},$$

one for each A_i. Each $freq_i$ maps P_i onto its frequency of occurrence as observed in the query history $\mathcal{Q}$.

For example, suppose that D is described by the single table $R(A_1, A_2)$, where A_1 ranges over the positive integers, and A_2 ranges over the finite set $\{v_1, \ldots, v_n\}$. Let the predicates found in the history of queries be:
$p_1 = (A_1 \in [10, 20])$
$p_2 = (A_1 \in [10, 20] \wedge A_2 = v_1)$
$p_3 = (A_1 \in [15, 30] \wedge A_2 = v_2)$
$p_4 = (A_2 \in \{v_1, v_2\})$
We write $[10, 20]$ as a shorthand for $A_1 \in [10, 20]$. The histograms are constructed as follows:

1. Q_1 carries predicate $[10, 20]$, so $freq_1([10, 20]) = 1$.
2. After Q_2, $freq_1([10, 20]) = 2$ and $freq_2(v_1) = 1$.
3. Q_3 causes the $[10, 20]$ interval to be *refined* into the adjacent disjoint intervals $[10, 15]$, $[15, 20]$ and $[20, 30]$, associating a frequency to each:
   $freq_1([10, 15]) = 2$,
   $freq_1([15, 20]) = 3$,
   $freq_1([20, 30]) = 1$
   (partitioning two overlapping intervals into disjoint intervals can be accomplished easily). Also, now $freq_2(v_1) = 2$ and $freq_2(v_2) = 1$.

The resulting histograms are illustrated in Figure 2 (a). Notice that these histograms are independent of each other: for simplicity, we do not account for the co-occurence of some of the predicates within the same query. Also, by construction the histograms only include the predicates that appear in the queries, rather than all possible combinations for the value set of each attribute.

Definition 2. *(Space of predicates)*
Given the sets $\{P_1, \ldots, P_l\}$ of predicate expressions for a request profile, the space of predicates $\mathcal{P}$ is the set of all vectors of the form $\mathbf{p} = (x_1, \ldots, x_l)$, with $x_k \in P_i$.

In the example, $\mathcal{P}$ includes $([10, 15], v_1)$, $([10, 15], v_2)$, $([15, 20], v_1)$ and so forth. We describe the popularity of a combination $\mathbf{p} \in \mathcal{P}$ of predicates using a syntectic score value:

$$score(\mathbf{p}) = \sum_{i:1..l} freq_i(p[i])$$

i.e., $score([15, 20], v_1) = 5$, $score([20, 30], v_2) = 2$, etc.

This score, however, is oblivious of the data from D_r that has already been purchased. Thus, it is complemented by the partial information on the completeness D relative to D_r:

Definition 3. *(Completeness map)*
Given $\mathbf{p} = (x_1, \ldots, x_l) \in \mathcal{P}$, let $p = x_1 \wedge \cdots \wedge x_l$ be a predicate. A completeness map is described by boolean function

$$missing(p) \in \{true, false\}$$

defined on the space of predicates, such that

$$missing(p) = true \text{ iff } Q_p(D_r) \subseteq D.$$

Assuming $missing(p) = true$ for all p initially, the map is updated using the history $\mathcal{Q}'$ of data purchases. For instance, let $p'_1 = (A_1 \in [15, 25] \wedge A_2 = v_1)$ be the predicate for Q'_1. As shown in Figure 2 (b), first p'_1 is used to further refine the histogram for A_1, adding the interval boundary 25. The corresponding *score* table is updated as a consequence. Splitting $[15, 25]$ into $[15, 20]$ and $[20, 25]$ aligns this interval with the existing histograms. Then, we set $missing([15, 20], v_1) = missing([20, 25], v_1) = 0$. In practice, we mark selected points in the space of predicates, which represent conjunctions that have already been used to purchase new data.

The rank of a point $\mathbf{p}$ representing missing data is simply the product

$$rank(\mathbf{p}) = score(\mathbf{p}) \ . \ missing(\mathbf{p})$$

This ranking provides a preference only among the predicates that describe popular items that are still missing (the others have rank 0), and replaces the simpler repair criterion used in the baseline version of our algorithm. In the example, the combination $([10, 15], v_1)$ is the most popular, since $([15, 20], v_1)$ is not missing.
It is worth mentioning that the operation of histogram refinement that may follow each user query, also requires reevaluating the *missing* function. This is simple, however, since each new sub-interval inherits the *missing* values found in the parent interval (this is left to intuition).

Algorithm INTENSIONAL DATA RANKING
Given relation $R(A_1, \ldots, A_l)$:

For user query $Q = \sigma_p(R)$:
begin
  for each A_i **do** {
    Let p_i be the conjunction of terms from p on A_i;
    affectedPoints = $refine(P_i, p_i)$;
    $updateHistogram(freq_i(p_i))$;
  }
  for each $\mathbf{p} \in$ affectedPoints **do** $updateScore(\mathbf{p})$;
  for each $\mathbf{p} \in \mathcal{P}$ **do** $rank(\mathbf{p}) = score(\mathbf{p}) \times missing(\mathbf{p})$;
end

For provider query $Q' = \sigma_{p'}(R)$ issued to D_r:
begin
  for each A_i **do** {
    Let p'_i be the conjunction of terms from p' on A_i;
    affectedPoints = $refine(P_i, p'_i)$;
    for each $\mathbf{p} \in$ affectedPoints **do** $updateScore(\mathbf{p})$;
    newPoints = $computeNewPoints(p'_i)$;
  for each $\mathbf{p} \in$ newPoints **do** $missing(\mathbf{p}) = false$;
end

Figure 3: Ranking algorithm

The ranking algorithm is summarized in Figure 3. Function $refine()$ increases the number of intervals in the histogram, and returns the points in the space of predicates that are affected by this operation. For these points, the score is updated prior to computing their rank. Upon issuing Q' to D_r, the provider must additionally compute the new points in the space of predicates corresponding to the query predicate, as shown earlier in the example, and reset their missing flag.

Tables *score* and *missing* can also be used as a basis to provide various estimates of completeness for a query $Q = \sigma_p$:

$$compl_{\sigma_p}(D, D_r) = \frac{|\sigma_p(D) \cap \sigma_p(D_r)|}{|\sigma_p(D_r)|}$$

For example, given predicate $p = [10, 20]$ with the situation illustrated in Figure 2, one may view intervals as discrete elements, regardless of their width, and observe that p corresponds to the "slice" of the *score* table that includes 4 predicate combinations. Since only one of these is not missing, one may estimate $compl_p(D, D_r) = 0.25$. Alternatively, the width of each of the intervals involved or other weight factors can be taken into account, yielding different estimates.

6. REFERENCE ARCHITECTURE

The architecture sketched in Figure 4 consists of a quality management module that contains the key components described in the previous sections. When a user query comes through the client service interface, it is processed as usual; before the result is returned, it is intercepted and passed to the quality management module, and query post-processing occurs. Using the interceptor pattern ensures that no changes are required to the query processor.

With reference to completeness, query post-processing proceeds as follows. The detection component is in charge of monitoring the completeness indicator and of computing the quality value, estimating the penalty associated with the query result. As explained in the previous section, this is done by querying the profile manager, which controls the completeness maps. The user query is also used by the profile manager to update the request profiles.

The strategy manager then uses this information to compute the utility function and to setup the optimization problem. Again, the profile manager is a critical component, in that it provides the ranked predicates that correspond to the most interesting new items. At this point, the repair actions consist of one or more queries to D_r, issued by the repair component using the pull interface of the gateway.

If the "instant repair" option is available, described in Section 4.1, then the new data items are prepared for immediate delivery and added to the original result set. Additionally, the queries are passed to the profile manager, which updates the completeness map.

If a push interface is active, then any update to D_r is propagated not only to D, but also to the profile manager, which may use it to update the completeness estimates. At the end of post-processing the final price is computed, taking self-assessed penalties into account, and the result is returned to the user. In the figure, an agreement interface is also shown as part of the quality management module, to indicate the channel used for agreement negotiation, which results in the configuration of the module components.

7. FURTHER WORK AND CONCLUSIONS

We have presented a simple model for the definition of formal agreements between data providers and consumers regarding the quality levels of data, illustrating the provider's problem space and showing an algorithm for dealing with the completeness property as a special case. Finally, we have described a reference provider architecture for enforcing quality agreements, that is respectful of the existing query processing architecture.

This work is at its initial stages and can be extended in many directions, adding elements that may affect our initial assessment of the agreement model and of the providers' strategies. Firstly, we believe that an experimental evaluation of the presented approaches for completeness, and a prototype implementation of the architecture, may provide an insight into their practicality. Secondly, the overall framework and architecture must be tested by considering additional properties: are there suitable architectural patterns for dealing with constraints on multiple quality properties, and how would the provider define strategies that involve interplay between properties, i.e., between completeness and currency? In the same vein, dealing with multiple agreements with overlapping constraints may intuitively make the provider's strategy more cost-effective, in ways that need to be investigated.

8. REFERENCES

[1] A.Borthwick, M.Buechi, and A.Goldberg. Key concepts in the choicemaker 2 record matching system. In *Procs. First Workshop on Data Cleaning, Record Linkage, and Object Consolidation, in conjunction with KDD 2003*, Washington, DC, July 2003.

[2] A.Dan, D.Davis, R.Kearney, A.Keller, R.King, D.Klueber, H.Ludwig, M.Polan, M.Spreitzer, and

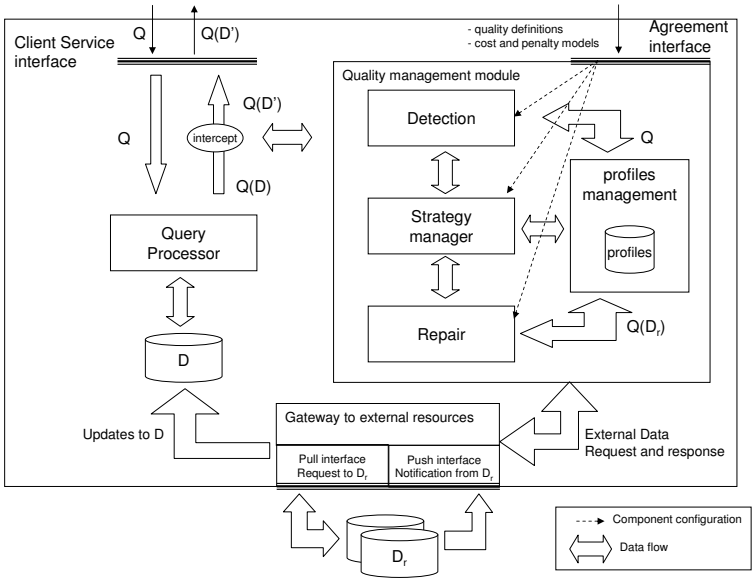

Figure 4: Reference architecture

A.Youssef. Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, 43(1), 2004.

[3] A.Motro, P.Anokhin, and A.C. Acar. Utility-based resolution of data inconsistencies. In Felix Naumann and Monica Scannapieco, editors, *International Workshop on Information Quality in Information Systems 2004 (IQIS'04)*, Paris, France, June 2004. ACM.

[4] C. Cappiello, C. Francalanci, P. Missier, B. Pernici, P. Plebani, M. Scannapieco, and A. Virgillito. Presentation of metadata and of the quality certificate. Deliverable dl2, The DaQuinCis project, 2003.

[5] D.Ballou and G.K.Tayi. Methodology for allocating resources for data quality enhancement. In *Communications of the ACM*, volume 32. ACM, March 1989.

[6] D.Ballou and H.Pazer. Designing information systems to optimize the accuracy-timeliness tradeoff. *Information Systems research*, 6(1), 1995.

[7] D.Ballou, R.Wang, H.Pazer, and G.K.Tayi. Modelling information manufacturing systems to determine information product quality. *Journal of Management Sciences*, 44(4), April 1998.

[8] M.G. Elfeky, A.K. Elmagarmid, and V.S. Verykios. Tailor: a record linkage tool box. In *Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)*, San Jose, CA, Feb. 2002. IEEE Computer Society.

[9] L. P. English. *Improving data warehouse and business information quality: methods for reducing costs and increasing profits*. John Wiley & Sons, 1 edition, March 1999. ISBN: 0471253839.

[10] I.P. Fellegi and A.B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64, 1969.

[11] F.Naumann, J.C.Freytag, and U.Leser. Completeness of integrated information sources. *Information Systems*, 29(7):583–615, 2004.

[12] S. Kalepu, S. Krishnaswarmy, and S.W. Loke. Verity: a qos metric for selecting web services and providers. In *Proceedings of 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03)*. IEEE Computer Society Press, 2004.

[13] P. Missier and C. Batini. A multidimensional model for information quality in cooperative information systems. In M. Helfert M. Eppler, editor, *Proceedings of the Eight International Conference on Information Quality (ICIQ-03)*, 2003.

[14] M.Lenzerini. Data integration: A theoretical perspective. In *Principles Of Database Systems*, pages 233–246, 2002.

[15] M.Scannapieco, A.Virgillito, C.Marchetti, M.Mecella, and R.Baldoni. The architecture: a platform for exchanging and improving data quality in cooperative information systems. *Inf. Syst.*, 29(7):551–582, 2004.

[16] M.Scannapieco and C.Batini. Completeness in the relational model: a comprehensive framework. In *Procs. 9th International Conference on Information Quality, ICIQ 2004, Cambridge, Ma*, 2004.

[17] F. Naumann, U.Leser, and J.C.Freytag. Quality-driven integration of heterogenous information systems. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, pages 447–458, Edinburgh, Scotland, UK, September 1999. Morgan Kaufmann.

[18] T.C. Redman. *Data quality for the information age*. Artech House, 1996.

[19] R.Y.Wang, M.Ziad, and Y.W.Lee. *Data quality*. Advances in Database Systems. Kluwer Academic Publishers, 2001.

[20] J. Skene, D. D.Lamanna, and W. Emmerich. Precise service level agreements. In *Proceedings of 26th International Conference on Software Engineering (ICSE'04)*. IEEE Computer Society Press, 2004.

[21] Y. Wand and R.Wang. Anchoring data quality

dimensions in ontological foundations. *Communications of the ACM*, 39(11), 1996.

[22] R. Wang. A product perspective on total data quality management. *Communications of the ACM*, 41(2), February 1998.

[23] R. Y. Wang, M. Ziad, and G. Shankaranarayanan. IP-MAP: representing the manufacture of an information product. In *Proceedings of the Eight International Conference on Information Quality (ICIQ-00)*, Cambridge, MA., November 2000.

[24] R.Y. Wang and D.M. Strong. Beyond accuracy: what data quality means to data consumers. *Journal of Management Information Systems*, 12(4), 1996.

Making Quality Count in Biological Data Sources

Alexandra Martinez
Dept of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611 USA
+1 (352) 392-1200
amartine@cise.ufl.edu

Joachim Hammer
Dept of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32611 USA
+1 (352) 392-1200
jhammer@cise.ufl.edu

ABSTRACT

We propose an extension to the semistructured data model that captures and integrates information about the quality of the stored data. Specifically, we describe the main challenges involved in measuring and representing data quality, and how we addressed them. These challenges include extending an existing data model to include quality metadata, identifying useful quality measures, and devising a way to compute and update the value of the quality measures as data is queried and updated. Although our approach can be generalized to various other domains, it is currently aimed at describing the quality of biological data sources. We illustrate the benefits of our model using several examples from biological databases.

1. INTRODUCTION

The rapid and continuous increase in the amount of biological information available (both experimental and derived) has brought concern over the perceived quality of existing data. Analysis and processing of faulty data produces misleading results and could hamper scientific progress. At the same time, most research efforts in the area of data quality assessment have been geared towards data quality problems in enterprise data warehousing, and less work has been aimed at analyzing and recording the data quality of biological databases. Hence there is a need for quality assessment measures in biological data sources and we propose a model for assessing and recording quality information in biological data sources.

We define data quality as a measure of the trustworthiness of the data. Data that is trustworthy conforms to reality, and therefore can be relied upon for any decision-making, analysis, or to derive new knowledge. Our definition of data quality is consistent with those that regard data quality as the degree of agreement between data views presented by an information system and that same data in the real-world [21]. However, the trustworthiness of the data is still a very abstract and broad concept, which is difficult to measure. In our approach we decided to decompose the notion of trustworthiness into different metrics each of which is a quantifiable value. In this sense, we also agree with the notion of data quality as a multi-dimensional concept [22, 26].

There has also been a growing interest from the database community in a special type of data known as *semistructured data* (or *unstructured data*), which is commonly defined as "schemaless" or "self-describing" data [1,5]. Semistructured data differs from the kind of data handled by relational databases in that it does not have a rigid structure; hence the interest in finding new ways of modeling and managing this type of data. The use of semistructured-like models in the biological context started off with the ACeDB system (which still requires a schema, but imposes only weak constraints) [1,5], and has recently gained popularity with the development of XML-based languages for biology such as BioML [25], BSML [4], AGAVE [2], and XEMBL [28], just to mention some. Currently, most biological repositories (including GenBank [10], EMBL [9], and DDBJ [8]) can export their data in XML format, using some of the existing XML languages. This trend suggests that semistructured data models (XML being one of them) are expressive and flexible enough to adequately represent biological data, which characterizes by having large variability and missing data.

The main contributions of our work are: 1) The definition of six quality measures that are meaningful in the biological domain; 2) the conceptual integration of these measures into a data model suitable for representing both biological data and quality measures; and 3) the description of how these quality measures change as data is queried or updated (according to the operations of our data model), and how they also affect the query result.

The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 presents our approach, particularly how we measure the quality of biological data, and how we integrate the quality information into a data model fitting both data and quality metadata. It also describes how the operations in the data model will change to encompass our quality metadata. Section 4 illustrates the use of our quality measures with several examples taken from RefSeq [20], a popular genomic data source. In Section 5 we outline our conclusions and suggestions for future work.

2. RELATED WORK

Research related to our work can roughly be divided into three areas: *Information Quality* research aimed at defining, measuring and evaluating the quality of data, *Data Quality* research in cooperative information systems, and research on *semistructured data models*. We briefly summarize the state-of-the-art in each area below.

IQIS 2005, June 17, 2005, Baltimore, MD, USA.

Information Quality (IQ) is commonly defined as "fitness for use" (i.e., achieving a level of data quality that is sufficient from the perspective of its users) [3]. Numerous models, evaluation methodologies, and improvement techniques have been developed in the area of Information Quality [12,13,16,24,27]. Wang et al. [27] proposed an attribute-based model to tag data with quality indicators, which characterize the data and its manufacturing process. They suggest a hierarchy of data quality dimensions with four major dimensions: accessibility, interpretability, usefulness, and believability. Each of these dimensions is split into other factors such as availability, relevancy, accuracy, credibility, consistency, completeness, timeliness, and volatility. Mihaila et al. [16] identified four Quality of Data parameters: completeness, recency, frequency of updates, and granularity. Lee et al. [12] distinguished five dimensions of data quality: accessibility, relevancy, timeliness, completeness, and accuracy; each considered a performance goal of the data production process. Lee et al. [13] developed a methodology for IQ assessments and benchmarks called AIM quality. This methodology is based on a set of intrinsic, contextual, representational, accessibility IQ dimensions which are important to information consumers. These dimensions were first devised by Strong et al. [24] as categories for high-quality data. All these research efforts offer valuable contributions for better understanding data quality problems and challenges, but they often lack the pragmatic component which will ultimately allow users to make decisions about the quality of the data based on quantitative measures.

More recently, Data Quality has been studied in the context of cooperative information systems [15,23,19,17], where more practical approaches have emerged. Mecella et al. [15] describe a service-based framework for managing data quality in cooperative information systems, based on an XML model for representing and exchanging data and data quality. Scannapieco et al. [23] developed the DaQuinCIS architecture and the D^2Q (Data and Data Quality) model for managing data quality in cooperative information systems. They defined four data quality dimensions: accuracy, completeness, currency, and consistency. Naumann et al. [19] presented a model for determining the completeness (i.e., a combination of density and coverage) of a source or combination of sources. Missier et al. [17] defined the notions of quality offer and quality demand within cooperative information systems, and modeled quality profiles as multidimensional date cubes. In spirit, these works are closely related to ours because they seek concrete and systematic means of computing the quality of data. However, our data quality model is not aimed for cooperative information systems but rather for single biological information systems. In the biological context, data quality has been addressed by Müller et al. [18], who examined the production process of genome data and identified common types of data errors.

In the area of semistructured data, several models have been proposed [7,5,1]. Most of them, however, use the same underlying representation: a graph-like or tree-like structure [5]. For example, Abiteboul et al. [1] use an edge-labeled graph to represent semistructured data. UnQL and LORE are based on an edge-labeled tree representation [6,14]. Calvanese et al. [7] use the basic data model for semi-structured data (called BDFS) in which both databases and schemas are represented as graphs. The work by Scannapieco et al. [23] is also relevant here because they provide a strong association between their D^2Q schemas and XML schemas, thus proving XML convenient for representing both data and data quality.

3. APPROACH

Our work addresses the lack of a formal model for measuring and representing quality information in biological data sources. Several challenges must be overcome. The first one is to identify a core set of quality measures which provides a framework for assessing the quality of biological data. A second challenge is to devise a data model for recording and maintaining biological data that facilitates the integration of quality measures into the model. A third challenge is to determine the effect of the operations defined in the data model on the quality measures, and the influence of the quality measures on the result of the operations. We address our approach in response to each of these challenges in the following subsections.

3.1 Measuring the Quality of Data

In order to identify useful quality measures, we define criteria that the measures should satisfy: *biologically-relevant*, *objective*, and *easy-to-compute*. Since our work is framed within a biological context, our measures should be meaningful to biologists so they can use these measures to discern among data of different quality levels. Additionally, our measures should be objective, meaning that there is no room for ambiguous interpretation (i.e., subjective appraisal) when assessing the value of a measure. Last but not least, our measures should be easy-to-compute in a real system that efficiently handles quality-augmented data in a scenario where data is constantly being updated.

Using the above constraints, we have identified the following six quality measures: Stability, Density, Time since Last Update, Redundancy, Correctness, and Usefulness. We have classified them into two sets: primary measures, and derived measures. Primary measures are independent of each other, while derived measures depend on one or more primary measures. We provide an intuitive description for each of them below. Formal definitions will be given shortly.

3.1.1 Primary Measures

- *Stability* indicates whether the data is undergoing a period of change. This measure is obtained by computing a weighted average of the magnitude of changes applied to a data item since its creation, where more weight is assigned to recent changes. Changes that happened long time ago will have a small weight and thus will not affect the current stability of the data item significantly. On the other hand, a recent change to the data item will have a large impact over its current stability value. A value of 1 corresponds to maximum stability while 0 corresponds to maximum instability.

- *Density* indicates the number of components (or attributes) that describe a data item. For instance, a data item that consists of a single value is less dense than a data item involving several values. Note that density does not take into account the number of bytes needed to store a data item, since each data item is considered a data unit regardless of the space it takes. The lowest density value a data item can have is 1, but there is no upper limit. Our density measure is comparable to the combination of density and coverage

described in [19] since we account for both the intension and extension of a data source. However, our density measure is formulated in a different way because we only consider the density of data items within a single source, as opposed to [19]'s mediator system which involves many sources and uses them to build a normalizing factor based on a universal relation.

- *Time since Last Update* (*TsLU*) measures the time elapsed since a data item was last updated. Note that TsLU is not the same as the age* of the data item. A TsLU value of 0 denotes recently updated data, whereas higher values denote elder data. Our TsLU measure differs from the currency quality dimension defined in [23] in that we do not assume knowledge about the time when a data value changes in the real world. Actually, in our biological context, real world data (such as DNA sequences) are supposed to remain unchanged; so all changes are due to a human curation process that aims to correct errors introduced during data *collection*.
- *Redundancy* measures the fraction of redundant information contained in a data item. A data item contains redundant information if any of its components (sub-items) is redundant. A data item is redundant if it represents the same real world object as any other data item in the data source. A redundancy value of 0 indicates the data item contains no redundant information, whereas a value close to 1 indicates that a large fraction of the information in the data item is redundant. Redundancy is related to the consistency quality dimension described in [23] in that redundant data items usually are also inconsistent (i.e., their data values conflict with each other). However, [23] only defines consistency for attribute values within the same schema element (entity) instance, but our redundancy measure is defined also across instances. In the biological context, redundancy can arise due to multiple submissions of the same "real world" data (e.g. a DNA or protein sequence) by different authors, which usually differ in their annotations.

3.1.2 Derived Measures

- *Correctness* is a measure of the accuracy of a data item. It can also be regarded as the degree of confidence that the data item represents true information. A correctness value of 1 means the data item is completely accurate (i.e., we are 100% confident that the data is true), while a correctness value of 0 means the data is wrong (i.e., we cannot have any confidence about the accuracy of the data). At an abstract level, our correctness measure expresses the same notion as the accuracy dimension described in [23], but it differs in the way it is computed. In [23], it is proposed to use a distance function between the value stored at the data source and the value considered as 'correct'. In biology, however, we cannot make the assumption that such 'correct' value is available, due to the uncertainty associated to the data collection process and the lack of understanding about many biological processes. Therefore, a different approach is needed to estimate the correctness value of biological data. The approach we propose is to use a combination of the stability and age of the data item.
- *Usefulness* measures the utility of a data item as a function of the density, correctness, and redundancy of its components. Particularly, the usefulness of a data item is the ratio between the amount of *information* provided by the data item and its total amount of data. The amount of *information* conveyed by a data item is the amount of non-redundant correct data contained in the data item. The total amount of data contained in a data item is the sum over the density of its components. A usefulness value of 1 means the data item is completely useful (i.e., all of its data is non-redundant and correct), while a value of 0 means the data item is useless.

We believe that the proposed derived measures provide the means for grasping important quality aspects of the data, while the simple primary measures serve to construct more complex quality features. Other measures can be obtained from different combinations of the primary and/or derived quality measures defined here. Therefore, our set of quality measures may be enlarged and improved if other measures are considered to be more meaningful in certain domains.

3.2 Integrating Quality Metadata and Data

By now we have a framework for measuring the quality of data along various dimensions. Our next challenge is the conceptual integration of these quality measures into a data model for validation purposes.

3.2.1 Choosing the right Data Model

We start by selecting a model to represent semistructured data, and we choose the graph model proposed by Abiteboul et al. [1]. The main reasons for choosing this data model are its flexibility (it can represent data lacking a fixed schema), its simplicity (syntactic constructions are simple and easy to understand), and its fitness for representing biological data (can handle data with missing values and high variability).

Roughly speaking, the model by Abiteboul et al. [1] is an edge-labeled directed graph that represents semistructured data expressions (called *ssd-expressions*). The following syntax taken from [1] describes how *ssd-expressions* can be constructed:

<*ssd-expr*> ::= <*value*> | oid <*value*> | oid
<*value*> ::= atomicvalue | <*complexvalue*>
<*complexvalue*> ::= {label : <*ssd-expr*>, …, label : <*ssd-expr*>}

Here, atomic values are either numbers or strings, and labels are strings of ASCII characters. Object identifiers (oid) are labels bound to the <*value*> that follows them, and we denote them with an ampersand prefix (e.g. &10 or &s1). We illustrate the use of this syntax in Figure 1, where a fragment of the RefSeq record NM_128079 is represented as an ssd-expression. Figure 2 shows the equivalent graph representation of the ssd-expression in Figure 1. The complete NM_128079 record in the original RefSeq format is shown in Figure 3. Next, we extend this data model to include our quality metadata.

* Age is the time elapsed since the data item was created.

```
{
  Locus : {
            accession : "NM_128079",
            length-bp : 1356,
            biomol : "mRNA",
            div : "PLN",
            update-date : "25-JAN-2005"
          },
  Definition : "Arabidopsis thaliana protein",
  Version : {
              id : "NM_128079.4",
              gi : 42569303
            },
  Source : {
            Taxname : "Arabidopsis thaliana",
            Organism : {
                         Orgname : {
                                     genus : "Arabidopsis",
                                     species : "thaliana"
                                   },
                         Lineage : "Eukaryota; Viridiplantae; Streptophyta…"
                        }
           }
}
```

Figure 1. Fragment of a RefSeq record represented with the syntax for *ssd-expressions*.

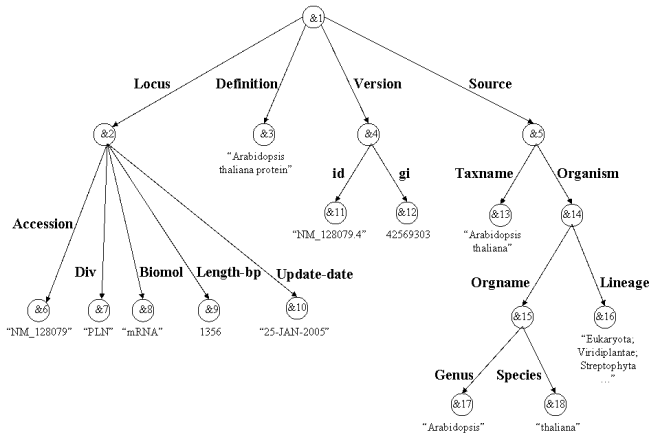

Figure 2. Edge-labeled graph representation of the RefSeq record from Figure 1.

3.2.2 Augmenting the Data Model with Quality Metadata

In order to incorporate our quality measures into the data model, we regard the set of all quality measures of a data item as its *quality metadata*. At a logical level, we represent *quality metadata* as a vector that has one entry (or dimension) per quality measure. Values stored in these entries are the scores associated to each of the quality measures; i.e.,

$$V_Q \equiv [Q_S, Q_D, Q_T, Q_R, Q_C, Q_U]$$

where Q_S, Q_C, Q_D, Q_T, Q_R, and Q_U are atomic objects that give a score for the quality measures *Stability*, *Density*, *TsLU*, *Redundancy*, *Correctness*, and *Usefulness,* respectively.

Using the syntax of our data model (*ssd-expressions*), *quality metadata* can be expressed as a special complex value having one label per quality measure and atomic values (measures scores) following the labels; i.e.,

$$\{S : Q_S, D : Q_D, T : Q_T, R : Q_R, C : Q_C, U : Q_U\}$$

where *S*, *D*, *T*, *R*, *C*, and *U* are labels that represent the *Stability*, *Density*, *TsLU*, *Redundancy*, *Correctness*, and *Usefulness* quality measures, respectively. Q_S, Q_D, Q_T, Q_R, Q_C, and Q_U are atomic values that confer a score to each quality measures. This way of representing the quality metadata is consistent with the data model.

The next issue we need to address is where to place the quality metadata associated to a data item. A data item denotes any of the following syntactic constructions: <*complexvalue*>, <*atomicvalue*>, or <*value*>; which would correspond to a node in the graph representation of the data model. Hence we could place a data item's quality metadata either at the data node or at the edge (i.e., label) coming into the data node. In what follows, we assume that the quality metadata is part of the data item's incoming edge. Making the quality metadata part of the edge (or label) entails some modifications to the original syntax from Section 3.2.1. The following extended syntax incorporates quality metadata into the data model for *ssd-expressions*:

<*ssd-expr*> ::= <*label*> : <*value*> | <*label*> : oid <*value*> |
<*label*> : oid

<*value*> ::= <*atomicvalue*> | <*complexvalue*>

<*atomicvalue*> ::= number | string

<*complexvalue*> ::= {<*ssd-expr*>, …, <*ssd-expr*>}

<*label*> ::= (labelstring, <*qmd*>)

<*qmd*> ::= {labelstring : number, …, labelstring : number}

Note that we have added a general construction for quality metadata (<*qmd*>), which resembles a complex value structure but has simplified labels and values. We have also added a rule for atomic values that allows us to distinguish between numbers and strings; which is needed in the <*qmd*> definition. We augmented the label's syntax so that it can hold quality metadata, and changed the placement of labels in the grammar so that every ssd-expression is forced to have a label rather than only the ssd-expressions within a complex value. Figure 4 shows how the RefSeq record from Figure 3 is represented using our quality-augmented syntax. Labels are merely illustrative: their names were created from the original record's field tags, and their quality metadata is symbolized with a letter 'q' (the reader should interpret this as the actual <*qmd*> structure $\{S : Q_S, D : Q_D, \ldots, U : Q_U\}$).

The semantics of our quality-augmented labels is as follows. If *l* is a label (or edge) with source node *s*(*l*) and target node *t*(*l*), then the quality metadata of *l*, denoted by *q*(*l*), pertain to the data represented by node *t*(*l*) and all its descendant nodes. If *t*(*l*) is a leaf node (i.e., atomic value), then *q*(*l*) refers to the quality metadata of the atomic value stored at that leaf. If *t*(*l*) is an internal node (i.e., complex value), then *q*(*l*) comprises the quality metadata of the complex value represented by the subtree rooted at that node.

```
LOCUS       NM_128079              1356 bp    mRNA    linear   PLN 25-JAN-2005
DEFINITION  Arabidopsis thaliana protein kinase family protein (At2g25220)
            mRNA, complete cds.
ACCESSION   NM_128079
VERSION     NM_128079.4  GI:42569303
KEYWORDS    .
SOURCE      Arabidopsis thaliana (thale cress)
  ORGANISM  Arabidopsis thaliana
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
            Spermatophyta; Magnoliophyta; eudicotyledons; core eudicots;
            rosids; eurosids II; Brassicales; Brassicaceae; Arabidopsis.
COMMENT     PROVISIONAL REFSEQ: This record has not yet been subject to final
            NCBI review. This record is derived from an annotated genomic
            sequence (NC_003071). The reference sequence was derived from
            mrna.At2g25220.1.
            On Feb 17, 2004 this sequence version replaced gi:30682693.
FEATURES             Location/Qualifiers
     source          1..1356
                     /organism="Arabidopsis thaliana"
                     /mol_type="mRNA"
                     /db_xref="taxon:3702"
                     /chromosome="2"
                     /map="unknown"
                     /clone="CHR2v01212004"
                     /ecotype="Columbia"
     gene            1..1356
                     /locus_tag="At2g25220"
                     /note="synonym: T22F11.19; protein kinase family protein"
                     /db_xref="GeneID:817060"
     CDS             1..1152
                     /locus_tag="At2g25220"
                     /note="contains protein kinase domain, Pfam:PF00069;
                     go_function: kinase activity [goid 0016301]"
                     /codon_start=1
                     /product="protein kinase family protein"
                     /protein_id="NP_180094.2"
                     /db_xref="GI:42569304"
                     /db_xref="GeneID:817060"
                     /translation="MGSGEEDRFDAHKKLLIGLIISFSSLGLIILFCFGFWVYRKNQS
                     PKSINNSDSESGNSFSLLMRRLGSIKTQRRTSIQKGYVQFFDIKTLEKATGGFKESSV
                     IGQGGFGCVYKGCLDNNVKAAVKKIENVSQEAKREFQNEVDLLSKIHHSNVISLLGSA
                     SEINSSFIVYELMEKGSLDEQLHGPSRGSALTWHMRMKIALDTARGLEYLHEHCRPPV
                     IHRDLKSSNILLDSSFNAKISDFGLAVSLDEHGKNNIKLSGTLGYVAPEYLLDGKLTD
                     KSDVYAFGVVLLELLLGRRPVEKLTPAQCQSLVTWAMPQLTDRSKLPNIVDAVIKDTM
                     DLKHLYQVAAMAVLCVQPEPSYRPLITDVLHSLVPLVPVELGGTLRLRTR"
ORIGIN
        1 atgggaagtg gtgaagaaga tagatttgat gctcataaga aacttctgat tggtctcata
       61 atcagtttct cttctcttgg ccttataatc ttgttctgtt ttggcttttg ggtttatcgc
      121 aagaaccaat ctccaaaatc catcaacaac tcagattctg agagtgggaa ttcattttcc
      181 ttgttaatga gacgacttgg ctcgattaaa actcagagaa gaacttctat ccaaaagggt
      241 tacgtgcaat ttttcgatat caagaccctc gagaaagcga caggcggttt taaagaaagt
      301 agtgtaatcg gacaaggcgg tttcggatgc gtttacaagg gttgtttgga caataacgtt
      361 aaagcagcgg tcaagaagat cgagaacgtt agccaagaag caaaacgaga atttcagaat
      421 gaagttgact tgttgagcaa gatccatcac tcgaacgtta tatcattgtt gggctctgca
      481 agcgaaatca actcgagttt catcgtttat gagcttatgg agaaaggatc attagatgaa
      541 cagttacatg ggccttctcg tggatcagct ctaacatggc acatgcgtat gaagattgct
      601 cttgatacag ctagaggact agagtatctc catgagcatt gtcgtccacc agttatccac
      661 agagatttga aatcttcgaa tattcttctt gattcttcct tcaacgccaa gatttcagat
      721 ttcggtcttg ctgtatcgct ggatgaacat ggcaagaaca acattaaact ctctgggaca
      781 cttggttatg ttgccccgga atacctcctt gacggaaaac tgacggataa gagtgatgtt
      841 tatgcatttg gggtagttct gcttgaactc ttgttgggta gacgaccagt tgaaaaatta
      901 actccagctc aatgccaatc tcttgtaact tgggcaatgc cacaacttac cgatagatcc
      961 aagcttccaa acattgtgga tgccgttata aaagatacaa tggatctcaa acacttatac
     1021 caggtagcag ccatggctgt gttgtgcgtg cagccagaac caagttaccg gccgttgata
     1081 accgatgttc ttcactcact tgttccactg gttccggtag agctaggagg gactctccgg
     1141 ttaacaagat gattcacaga aacacgccaa aagaaatcca aagccattta gatgattttc
     1201 ttttatcctt tgcctttata ttttttttgta tagggttatg atccactcat ctgaaagttt
     1261 gggggtaaga atgtgagaat ataagttttc agggttgttg agttctatat aattatattt
     1321 gtttcttttt attgtcaaat ataattatat ttttgt
//
```

Figure 3. Record with Accession number NM_128079.4, taken from RefSeq.

```
(Seq-entry,q):{
        (LOCUS,q):{
                (accession,q):"NM_128079",
                (length-bp,q):1356,
                (biomol,q):"mRNA",
                (div,q):"PLN",
                (update-date,q):"25-JAN-2005"
        }
        (DEFINITION,q):"Arabidopsis thaliana protein kinase family protein (At2g25220) mRNA, complete cds."
        (ACCESSION,q):"NM_128079"
        (VERSION,q):{
                (id,q):"NM_128079.4",
                (gi,q):42569303
        }
        (KEYWORDS,q):""    .
        (SOURCE,q):{
                (taxname,q):"Arabidopsis thaliana (thale cress)",
                (organism,q):{
                        (orgname,q):{
                                (genus,q):"Arabidopsis",
                                (species,q):"thaliana"
                        }
                        (lineage,q):"Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;
                                Spermatophyta; Magnoliophyta; eudicotyledons; core eudicots;
                                rosids; eurosids II; Brassicales; Brassicaceae; Arabidopsis."
                }
        }
        (COMMENT,q):"PROVISIONAL REFSEQ: This record has not yet been subject to final
            NCBI review. This record is derived from an annotated genomic
            sequence (NC_003071). The reference sequence was derived from
            mrna.At2g25220.1.
            On Feb 17, 2004 this sequence version replaced gi:30682693."
        (FEATURES,q):{
                (source,q):{
                        (location,q):{
                                (from,q):1,
                                (to,q):1356
                        }
                        (qualifiers,q):{
                                (organism,q):"Arabidopsis thaliana",
                                (mol_type,q):"mRNA",
                                (db_xref,q):"taxon:3702",
                                (chromosome,q):"2",
                                (map,q):"unknown",
                                (clone,q):"CHR2v01212004",
                                (ecotype,q):"Columbia"
                        }
                }
                (gene,q):{
                        (location,q):{
                                (from,q):1,
                                (to,q):1356
                        }
                        (qualifiers,q):{
                                (locus_tag,q):"At2g25220",
                                (note,q):"synonym: T22F11.19; protein kinase family protein",
                                (db_xref,q):"GeneID:817060"
                        }
                }
                (CDS,q):{
                        (location,q):{
                                (from,q):1,
                                (to,q):1152
                        }
                        (qualifiers,q):{
                                (locus_tag,q):"At2g25220",
                                (note,q):"contains protein kinase domain, Pfam:PF00069; go_function: kinase activity [goid 0016301]",
                                (codon_start,q):1
                                (product,q):"protein kinase family protein",
                                (protein_id,q):"NP_180094.2",
                                (db_xref,q):"GI:42569304",
                                (db_xref,q):"GeneID:817060",
                                (translation,q):"MGSGEEDRFDAHKKLLIGLIISFSSLGLIILFCFGFWVYRKNQS
                                        PKSINNSDSESGNSFSLLMRRLGSIKTQRRTSIQKGYVQFFDIKTLEKATGGFKESSV
                                        IGQGGFGCVYKGCLDNNVKAAVKKIENVSQEAKREFQNEVDLLSKIHHSNVISLLGSA
                                        SEINSSFIVYELMEKGSLDEQLHGPSRGSALTWHMRMKIALDTARGLEYLHEHCRPPV
                                        IHRDLKSSNILLDSSFNAKISDFGLAVSLDEHGKNNIKLSGTLGYVAPEYLLDGKLTD
                                        KSDVYAFGVVLLELLLGRRPVEKLTPAQCQSLVTWAMPQLTDRSKLPNIVDAVIKDTM
                                        DLKHLYQVAAMAVLCVQPEPSYRPLITDVLHSLVPLVPVELGGTLRLTR"
                        }
                }
        }
        (ORIGIN,q):  "1 atgggaagtg gtgaagaaga tagatttgat gctcataaga aacttctgat tggtctcata
                        ...
                        1321 gtttcttttt attgtcaaat ataattatat ttttgt"
}
```

Figure 4. Representation of the RefSeq record NM_128079.4 (in Figure 3) using our quality-augmented semistructured data model.

3.2.3 Computing the Score of the Quality Measures

We are left now with the issue of how to effectively compute the score of each of our quality measures for a particular data item. We propose a set of heuristic formulae, which are largely based on historical information about the data (i.e., what was the value of the data in previous versions, how much time elapsed between versions, when was the last time it was updated, etc.). After analyzing some records from RefSeq together with their "revision history", we recognized the potential application of this historical data to the computation of our quality measures. Extracting the appropriate information from the different versions that a data item has undergone over time can reveal quality aspects such as stability, correctness, and currency of the data item at hand. We do not claim that our measure scores are optimal; in fact, there may even be more effective ways of computing the scores, so consider our formulae just as a starting point for future discussions on this subject.

For simplicity, most of the formulae we present here assume that the data graph is acyclic. However, they can be extended to account for cycles in the graph. In particular, we will assign scores to each of the quality measures in the quality metadata $q(l)$ of a label l as follows.

(i) If the label's target node $t(l)$ is an atomic value v, then:

- The **Stability** score Q_S in $q(l)$ is defined as

$$Q_S = 1 - \sum_{i=1}^{n} [\Delta(v(i-1), v(i)) \times \int_{t_i}^{t_{i-1}} \lambda e^{-\lambda t} dt] \qquad (1)$$

  where n is the number of versions v has undergone, t_i is the time elapsed since the ith version of v ($t_0 \equiv \infty$), and $v(i)$ is the state[2] of v at version i. Δ is a function that measures the proportion of change (if you want, the "distance") between two atomic values, which we will shortly define. The integral in formula (1) weights the changes depending on when they occur in time: recent changes have more weight than old changes; here $\lambda > 0$ is a parameter to be determined experimentally.

- The **Density** score Q_D in $q(l)$ is defined as

$$Q_D = 1 \qquad (2)$$

  which means that each atomic value counts as one data unit.

- The **TsLU** score Q_T in $q(l)$ is defined as

$$Q_T = \log\left(1 + \left\lfloor \frac{t-u}{f} \right\rfloor\right) \qquad (3)$$

  where t is the current time, u is the time when v was last updated, and f is the frequency of update[3] of the database. We use a log-based scale for time because we consider that it is fairer to convert time to a logarithmic scale when making comparisons.

- The **Redundancy** score Q_R in $q(l)$ is defined as

$$Q_R = \frac{|T_v| - 1}{|T_v|} \qquad (4)$$

  where T_v is the set of ssd-expressions in the database that are redundant with respect to v, and $|T_v|$ is the cardinality of this set. We assume that v itself is included in T_v so that if v is unique then $|T_v|=1$ and $Q_R=0$, and if v is redundant then $|T_v|>1$ and $Q_R>0$. In order to determine which data items are redundant one could use, for instance, the algorithms described in [11].

- The **Correctness** score Q_C in $q(l)$ is defined as

$$Q_C = w_1 \times Q_S + w_2 \times (1 - e^{-\beta \times age}) \qquad (5)$$

  where $\beta > 0$, $0 \le w_1 \le 1$, and $w_2 = 1\text{-}w_1$, are parameters to be determined experimentally. Q_S is the stability score, and *age* is the time elapsed since the creation of v. In other words, correctness is a weighted average of the stability and the age, with age being first mapped to the interval [0,1) (0 meaning that the data item is new, and a value close to 1 meaning that the data item is very old).

- The **Usefulness** score Q_U in $q(l)$ is defined as

$$Q_U = q(l).D \times q(l).C \times [1 - q(l).R] \qquad (6)$$

  where $q(l).D$, $q(l).C$, and $q(l).R$ are path expressions for the density, correctness, and redundancy scores associated to v.

We define the function $\Delta(v_1,v_2)$ for atomic values v_1 and v_2 as follows:

If v_1 and v_2 are of different type, then

$$\Delta(v_1, v_2) = 1$$

If v_1, v_2 are both strings, then

$$\Delta(v_1, v_2) = \frac{editDist\ (v_1, v_2)}{\max(length(v_1), length(v_2))}$$

If v_1, v_2 are both numbers, then

$$\Delta(v_1, v_2) = \frac{|v_2 - v_1|}{\max(v_1, v_2)}$$

(This formula assumes that v_1, v_2 are positive numbers.)

In any other case,

$$\Delta(v_1, v_2) = 1$$

Note that $0 \le \Delta(v_1, v_2) \le 1$ for any data pair (v_1, v_2). The aim of the Δ function is to assess the proportion of data that changed from value v_1 to value v_2. It can also be interpreted as a normalized distance between values v_1 and v_2. For example, if v_1 and v_2 are equal, then we need not change v_1 to obtain v_2, so $\Delta(v_1,v_2) = 0$. On the other hand, if v_1 and v_2 are strings and they differ in 50% of their content (i.e., 50% of v_1 needs to change so that v_1 becomes v_2), then $\Delta(v_1,v_2) = 0.5$.

[2] The state of an object is a given by its type and contents.

[3] Frequency of update indicates how often the database gets updated (e.g. daily, weekly).

(ii) If the label's target node $t(l)$ is a complex value $v = \{l_1 : e_1, \ldots, l_n : e_n\}$, then:

- The **Stability** score Q_S in $q(l)$ is defined as

$$Q_S = \frac{1}{n}\sum_{i=1}^{n} q(l_i).S \quad (7)$$

where n is the number of labels contained in v (i.e., outgoing edges from node $t(l)$), l_i is the i-th label in v, and $q(l_i).S$ is a path expression for the stability score associated to e_i (where e_i is a child node of $t(l)$).

- The **Density** score Q_D in $q(l)$ is defined as

$$Q_D = 1 + \sum_{i=1}^{n} q(l_i).D \quad (8)$$

where n and l_i are defined as in (7), and $q(l_i).D$ is a path expression for the density score of e_i. Q_D is in fact the number of nodes in the subtree whose root is v, i.e., the number of descendants of node v. We add one to this quantity to account for the node v itself. Formula (8) assumes that our data graph is effectively a tree (i.e., every node has only one parent node or incoming edge).

- The **TsLU** score Q_T in $q(l)$ is defined as

$$Q_T = \frac{1}{n}\sum_{i=1}^{n} q(l_i).T \quad (9)$$

where n and l_i are defined as in (7), and $q(l_i).T$ is a path expression for the TsLU score of e_i. Since the time for atomic values is given in a logarithmic scale, Q_T will not be too sensitive to extremely old data.

- The **Redundancy** score Q_R in $q(l)$ is defined as

$$Q_R = \frac{1}{n}\sum_{i=1}^{n} q(l_i).R \quad (10)$$

where n and l_i are defined as in (7), and $q(l_i).R$ is a path expression for the redundancy score of e_i.

- The **Correctness** score Q_C in $q(l)$ is defined as

$$Q_C = \frac{1}{n}\sum_{i=1}^{n} q(l_i).C \quad (11)$$

where n and l_i are defined as in (7), and $q(l_i).C$ is a path expression for the correctness score associated to e_i.

- The **Usefulness** score Q_U in $q(l)$ is defined as

$$Q_U = \frac{\sum_{i=1}^{n} q(l_i).D \times q(l_i).C \times \left[1 - q(l_i).R\right]}{\sum_{i=1}^{n} q(l_i).D} \quad (12)$$

where n and l_i are defined as in (7), and $q(l_i).D$, $q(l_i).C$, and $q(l_i).R$ are path expressions for the density, correctness, and redundancy scores of e_i, respectively.

At this point we can justify our formulae based on our initial experience with biological data, but we are in the process of validating this model with the development of a prototype and through posterior feedback from biologists.

3.3 Updating the Quality Measures under the Data Operations

Since we are primarily concerned about biological data, we cannot make the assumption that our data is mainly static. Conversely, we must consider a scenario where data is constantly updated through the operations defined in the data model. Thus, we need to address the issue of how our quality metadata is affected by each of the operations in the data model. For this purpose, we will consider a core set of operations on the graph model, which includes navigation, insertion, update, and deletion.

3.3.1 Navigating to a node n and returning its content

Let l be the last label (or edge) in the path through which we reached n, i.e., $t(l) = n$. None of the scores of the quality measures change under this operation. Together with the data represented by node n, this operation will return $q(l)$ as the quality metadata of the result.

3.3.2 Inserting a new node n

Let $l_1, l_2, \ldots, l_k$ be the path where n will be inserted, i.e. n will become a child of $t(l_k)$. Then, when the node is inserted we will assign initial scores to the quality measures of n. These initial scores are given by the formulae from previous section.

If the node to insert is a leaf (atomic value),

(a) Compute the quality measures for this leaf node according to formulas (1) through (6).

If the node to insert is an interior node (complex value),

(a) First, recursively compute the quality measures for its children (direct descendants).

(b) Then, compute the quality measures for this interior node according to formulas (7) through (12).

In either case, update the quality measures of all the nodes in the path from the root to n i.e., $t(l_1)$, $t(l_2)$,…, $t(l_k)$ so that they incorporate information about their new descendant. More specifically, this can be done using the procedure of an Update operation (see below).

Besides the path to the recently inserted node n, this operation will return the quality metadata $q(l_k)$ assigned to this new node.

3.3.3 Updating a node n

Let $l_1, l_2, \ldots, l_k$ be the path where n is located in the graph so that n is a child of $t(l_k)$. Then, when the node is updated, the scores of the quality measures will change according to the formulae from previous section.

If the node to update is a leaf (atomic value),

(a) Recompute the quality measures of this leaf node as specified by formulas (1) through (6).

If the node to update is an interior node (complex value),

(a) Recursively recompute the quality measures of those children that were affected by the update operation.

(b) Recompute the quality measures of this node as specified by formulas (7) through (12).

In either case, update the quality measures of all the nodes along the path from the root to n, i.e., $t(l_1)$, $t(l_2)$,…, $t(l_k)$.

This operation will return the updated quality metadata $q(l_k)$ of the just updated node n.

3.3.4 *Deleting a node n*

Let l_1, l_2, …, l_k be the path where n is located in the graph so that n is the child of $t(l_k)$. Then, when the node is deleted, the scores of the quality measures will be updated according to the formulae from previous section.

If the node to delete is a leaf (atomic value), then

(a) Delete label l_k (which contains the quality metadata for n).

(b) Recompute the quality measures of n's parent node, $t(l_{k-1})$, as specified by formulas (7) through (12).

If the node to delete is an interior node (complex value), we need to distinguish between two cases: (i) single node deletion, and (ii) subtree deletion. Hence,

(i) Single node delete (in this case, re-structuring of the tree is needed)

(a) Replace label l_k with the set of labels directly reached by l_k so that n's parent node becomes the parent of n's children nodes.

(b) Recompute the quality measures of n's parent node, $t(l_{k-1})$, as specified by formulas (7) through (12).

(ii) Subtree delete (no re-structuring of the tree)

(a) Delete label l_k (which contains the quality metadata for n) and all labels reachable from l_k in the subtree (which contain the quality metadata for the descendants of n).

(b) Update the quality measures of all the nodes in the path from the root to $t(l_{k-1})$ i.e., $t(e_1)$, $t(e_2)$, …, $t(e_{k-1})$) to account for the deletion of their descendants.

This operation will return the updated quality metadata $q(l_{k-1})$ of the just removed node's parent.

4. EXAMPLES

In this section, we illustrate the benefits of our model using several examples from biological databases.

Example 1. Suppose a biologist wants to have an estimate of the quality of the atomic value whose path expression is *Locus.Length-bp* (oid &9) in Figure 2. By using our quality-augmented data model, she obtains the following quality metadata for the atomic value 1356:

$$\{S: 0.9810, D: 1, T: 2.5366, R: 0, C: 0.9248, U: 0.9248\}$$

These scores were calculated using the twelve existing versions of the RefSeq record NM_128079 (as of March 2005), with parameters values $\lambda = 0.007$, $\beta = 0.002$, $w_1 = 0.4$ and $w_2 = 0.6$. The current time t in formula (4) was assumed to be date when the last version of the record occurred (January 25th, 2005), and f, the frequency of update of the database, was assumed to be 1 (daily updated). Also, we use a redundancy score of 0 for all our example values since RefSeq is by nature a non-redundant curated database.

The quality metadata shown above would be located at the edge (label) *Length-bp* in our model. Careful examination of these quality scores can provide greater insights into different quality aspects of the data at hand. For instance, we can immediately note that the current value of our atomic data was last updated 2.5366 log-time units ago (see *T* score). We can also know that this value is not redundant (see *R* score). Furthermore, we can deduce that this atomic value has had minimal or no changes during the most recent versions because its stability score is close to 1 (see *S* score). We can also be 92.48% confident about the accuracy of this atomic value (see *C* score). Based on the density score, we can infer that this atomic value conveys just one unit of data. The usefulness score suggests that 92.48% of the data in this atomic value is non-redundant and correct.

Figure 5 shows the behavior of the stability and correctness quality measures over time for the atomic value from Example 1. Time 0 corresponds to the date when this value was first inserted into the data source, and subsequent time markers correspond to the time elapsed since then, in years. Along with the quality scores, we also plot the magnitude of change from one version to the next one, which is defined by the Δ function. From Figure 5 we observe that whenever there is a change in the contents of our atomic value, the stability score either decreases or does not increase significantly with respect to its value on the previous version. Similarly, the correctness score is affected by changes to the value but it is not as sensitive to them as stability because it considers also the age of the data value. To exemplify this, observe how the scores for stability and correctness change from the version at time 1.6 to the version at time 2.05. The stability score drops from 0.91 to 0.79 due to a change of 27% in the contents of the value, whereas the correctness score remains at 0.78. This happens because the age of the value became significant, making correctness more robust to changes.

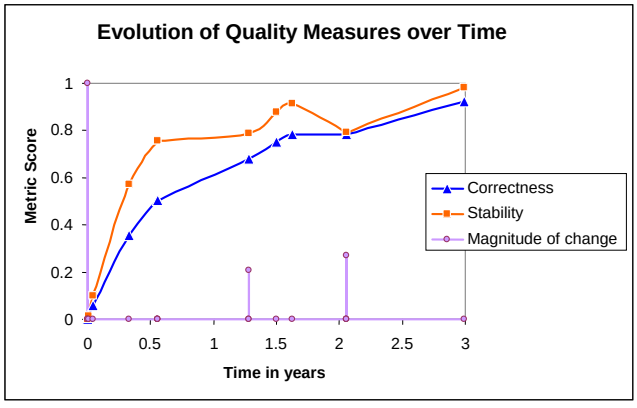

Figure 5. Evolution of Correctness and Stability scores over time for the atomic value at path *Locus.Length-bp*.

Example 2. Suppose the same biologist from Example 1 wants to have an estimate of the quality of the atomic value whose path expression is *Locus.Biomol* (oid &8) in Figure 2. By using our quality-augmented data model, she obtains the following quality metadata for the atomic value "mRNA":

$$\{S: 0.9995, D: 1, T: 3.0386, R: 0.5, C: 0.9322, U: 0.4661\}$$

These scores were calculated using the same parameters and settings as in Example 1. The quality metadata shown above is located at the edge (label) *Biomol* in our model. Let's examine these quality scores. We can see that the current contents of our atomic value were last updated 3.0386 log-time units ago, which suggest that this value has remained valid for longer time than the value from Example 1. We also note that this value is as dense as the value from Example 1 because each of them encloses one unit of data. It is also observed that the stability score is very close to 1, which implies that there were probably no significant changes to the contents of the value during the most recent versions. By comparing the stability score in this example and in Example 1, we deduce that this atomic value is currently more stable (i.e. has suffered less recent changes) than the value from Example 1. The usefulness score indicates that only 46.61% of the data in this atomic value conveys information (non-redundant correct data); thus assessing lower quality to this value than to the value in Example 1.

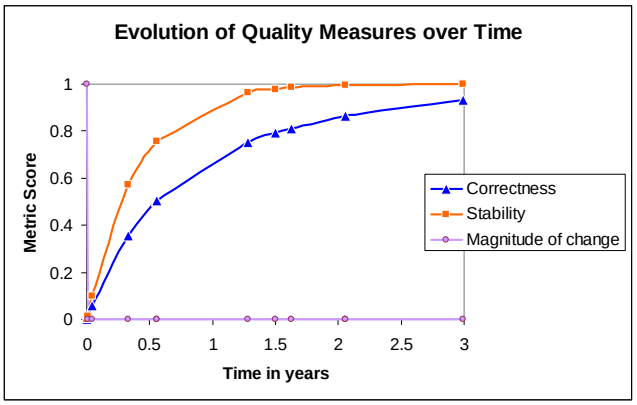

Figure 6. Evolution of Correctness and Stability scores over time for the atomic value at path *Locus.Biomol*.

Figure 6 shows the behavior of the correctness and stability quality measures over time for the atomic value *Locus.Biomol*. As before, time 0 corresponds to the date when this value was first inserted into the data source. Together with the quality scores, we plot the magnitude of change (defined by the Δ function) from one version to the next one. Note that the only change for this atomic value occurred when it was inserted. The curves for stability and correctness in Figure 6 show the typical convergence of these measures to their maximum values.

Example 3. Suppose our biologist from Examples 1 and 2 wants to have an estimate of the quality of the atomic value whose path expression is *Locus.Update-date* (oid &10) in Figure 2. By using our quality-augmented data model, she obtains the following quality metadata for the atomic value "25-JAN-2005":

$$\{S: 0.4676, D: 1, T: 0, R: 0.75, C: 0.7195, U: 0.1799\}$$

These scores were calculated using the twelve versions of the record NM_128079, and the same parameters from Example 1. The quality metadata shown above is located at the edge (label) *Update-date* in our model. The update-date value is not a very interesting data to analyze by itself since we know that it will change in every version and will simply contain the date when the last version took place. However, we use it here to illustrate the behavior of a value that changes frequently (as opposed to the data in Example 2, which does not change after insertion). Let's then examine the quality scores of this atomic value. The TsLU score of our atomic value is 0, which means that this value was updated during the last version and that the 'current time' is close to the time when this last version occurred. Note that the stability and correctness scores are significantly lower than the scores obtained in previous examples. In particular, a low stability score such as 0.4676 indicates the presence of large and recent changes in this atomic value (see Figure 7). The low correctness score is also an accumulated effect of the many changes experienced by this value over time, but it is especially influenced by the most recent changes. The correctness score of this atomic value is larger than the stability score because the age of the value is also considered. The usefulness score suggests that the fraction of non-redundant correct data conveyed by this atomic value is just 17.99% (lower than in Examples 1 and 2).

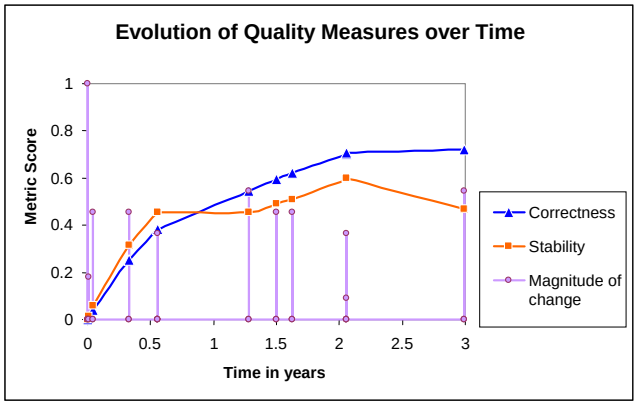

Figure 7. Evolution of Correctness and Stability scores over time for the atomic value at path *Locus.Update-date*.

Figure 7 shows the behavior of the correctness and stability quality measures over time for the atomic value *Locus.Update-date*. Observe that the relative change for this atomic value at each version is considerable (more than 0.3 or 30%). The stability and correctness scores from Figure 7 are far from reaching the maximum possible value (i.e., 1), and they will not increase much if this rate of change continues.

Example 4. Suppose our biologist from Examples 1, 2, and 3 needs to estimate the quality of the complex value whose path expression is *Locus* (oid &2) in Figure 2. By using our quality-augmented data model, she obtains the following quality metadata for the complex data at hand:

$$\{S: 0.8894, D: 6, T: 2.3305, R: 0.35, C: 0.8882, U: 0.6871\}$$

These scores were calculated using the same parameters as in Example 1. The quality metadata shown above is located at the edge (label) *Locus* in our model. We observe that the current

contents of our complex value were updated 2.3305 log-time units ago. We also note that 35% of the data in this complex value is redundant. The stability score of this complex value indicates that the average stability of its components is reasonably high. Likewise, the correctness score of this complex value suggests that the accuracy of its components, in average, is relatively high. A density score of 6 tells us that our complex value contain more data units (i.e., it is more dense) than the single atomic values from previous examples. We can infer from the usefulness score that 68.71% of the data contained in this complex value is accurate and non-redundant.

5. FUTURE WORK

We have proposed a quality-aware data model for representing and integrating quality metadata into a data source. Our model is tailored to biological data sources where ongoing concern over the low quality of rapidly growing experiment data has spurned the desire for quality assessment measures. We approached this problem in three steps.

First, we identified a set of quality measures relevant in a biological domain (but these measures could be easily extended to other domains). We give a clear and formal definition for each of the quality measures, specifying how to compute their score. Second, we selected a suitable data model and augmented it by incorporating the quality measures in a consistent manner. Third, we described how the quality measures are affected by each of the operations in the data model, and how the quality measures extend the result of the operations.

We plan on continuing the development of a prototype of our quality-aware model, which would allow us to validate the overall approach and demonstrate that the proposed quality measures are capable of providing meaningful and valuable information. An important part of the prototype is to implement the data operations in an efficient way so that updates are incremental. We also plan to explore ways for automatically capturing the data quality of existing data in biological repositories. This would enable a smooth migration from current to quality-aware data sources.

6. ACKNOWLEDGMENTS

Special thanks to Arturo Camacho for his useful discussions and reviews.

7. REFERENCES

[1] Abiteboul, S., Buneman P., Suciu, D. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, 2000.

[2] AGAVE - Architecture for Genomic Annotation, Visualization and Exchange. Available at http://www.agavexml.org/

[3] Ballou, D., Madnick, S., and Wang, R. Assuring Information Quality. *Journal of Management Information Systems*, *20*, 3(2004), 9-11.

[4] BSML -Bio Sequence Markup Language. Available at http://www.bsml.org/

[5] Buneman, P. Semistructured Data. *Proc. PODS '97.* Tucson, Arizona (May 1997).

[6] Buneman, S., Davison, S., Hillebrand, G., and Suciu, D. A query language and optimization techniques for unstructured data. *Proceedings of the ACM SIGMOD International Conference on Management of Data.* (1996), 505-516.

[7] Calvanese, D., De Giacomo, G., and Lenzerini, M. Modeling and Querying Semi-Structured Data. *Networking and Information Systems Journal*, *2*, 2(1999), 253-273.

[8] DDBJ -DNA Data Bank of Japan. Available at http://www.ddbj.nig.ac.jp/

[9] EMBL Nucleotide Sequence Database. Available at http://www.ebi.ac.uk/embl/

[10] GenBank. Available at http://www.ncbi.nlm.nih.gov/Genbank/index.html

[11] Hammer, J. and Pluempitiwiriyawej, C. Element matching across xml sources using a multi-strategy clustering technique. *Data and Knowledge Engineering (DKE), Elsevier Science*, *48* (2004), 297-333.

[12] Lee, Y.W. and Strong, D. M. Knowing-Why About Data Processes and Data Quality. *Journal of Management Information Systems, 20,* 3 (Winter 2003-4), 13-39.

[13] Lee, Y.W., Strong, D. M., Kahn, B.K., and Wang, R.Y. AIMQ: A methodology for information quality assessment. *Information & Management, 40,* 2(2002), 133-146.

[14] McHug, J., Abiteboul, S., Goldman, R., Quass, D., and Widom, J. Lore: A database management system for semistructured data. *SIGMOD Record*, *26*, 3(1997).

[15] Mecella, M., Scannapieco, M., Virgillito, A., Baldoni, R., Catarci, T., Batini, C. Managing Data Quality in Cooperative Information Systems. *Journal of Data Semantics*, *I* (2003), LNCS 2800.

[16] Mihaila, G., Raschid, L., Vidal, M. E. Querying "quality of data" metadata. *Proc. of the Third IEEE Meta-Data Conference.* Bethesda, Maryland (April 1999), 526-531.

[17] Missier, P., Batini, C. A Multidimensional Model for Information Quality in Cooperative Information Systems. *Proceedings of the Eighth International Conference on Information Quality* (2003), 25-40.

[18] Müller, H., Naumann, F., Freytag J.C. Data Quality in Genome Databases. *Proceedings of the Eighth International Conference on Information Quality* (2003), 269-284.

[19] Naumann, F., Freytag J.C., Leser, U. Completeness of integrated information sources. *Information Systems*, *29*, 7(2004), 583-615.

[20] NCBI Reference Sequences. Available at http://www.ncbi.nlm.nih.gov/RefSeq/

[21] Orr, K. Data Quality and Systems Theory. *Communications of the ACM, 41,* 2(1998), 66-71.

[22] Pipino, L.L., Lee, Y. W., and Wang, R. Y. Data Quality Assessment. *Communications of the ACM, 45,* 4(2002), 211-218.

[23] Scannapieco, M., Virgillito, A., Marchetti, M., Mecella, M., Baldoni, R. The DaQuinCIS Architecture: a Platform for Exchanging and Improving Data Quality in Cooperative

Information Systems. *Information Systems*, *29*, 7(2004), 551-582.

[24] Strong, D., Lee, Y., and Wang, R. Data quality in context. *Communications of the ACM, 40*, 5(1997), 103-110.

[25] The Biopolymer Markup Language –BIOML, Working Draft Proposal. Available at http://www.proteome.ca/x-bang/bioml/b_toc.htm

[26] Wand, Y. and Wang, R. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM, 39*, 11(1996), 86-95.

[27] Wang, R.Y., Reddy, M. P., and Kon, H.B. Toward quality data: An attribute-based approach. *Decision Support Systems, 13* (1995), 349-372.

[28] XEMBL. Available at http://www.ebi.ac.uk/xembl/

IQIS 2005, June 17, 2005, Baltimore, MD, USA.

Mapping this grand view to concrete technical choices requires the tuning of several components of the architecture. Following, we quickly summarize our findings that affected our architectural choices.

Starting with the sources, in this paper, we have focused on legacy systems. Apart from the requirement of minimal changes at the source side, legacy sources pose the interesting problem of having an application (instead of a DBMS) managing the data. We modify a library of routines for the management of data to allow the interception of the calls without affecting the applications. The modification involves (a) inserting no more than 100 lines of code to a library of routines for source management and (b) recompiling the application (which was not affected), over this library. Also, as far as the communication between stages is concerned, we transmit blocks of records for reasons of performance and minimal overhead of the source system.

The internal architecture of the intermediate layer (ADSA) is not obvious, either. For each ETL activity, we employ a queue to store incoming records before they are processed. Each activity processes the incoming data on-line and then passes its output to the next queue for further processing. Again, for reasons of performance, the unit of exchange is blocks of records and not individual records. We do not assume a fixed set of ETL operators, but rather we provide a taxonomy of such operations, based on their operational semantics. New operators can be added to the taxonomy as they are defined. To predict the performance of the system, we employ queue theory for networks of queues (cf. section 2.1 for a reminder of queue theory). Our experimental results indicate that the assumption of an M/M/1 queue for each of the ETL activities provides a successful estimation.

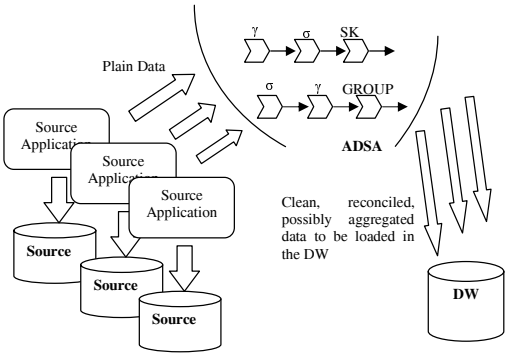

Fig. 1. Architecture Overview

To implement the requirement for stable interface at the side of the warehouse, the data are further propagated towards the warehouse through an interface involving web services [2]. The need for web services as the technical solution for populating the warehouse with fresh data is not self-evident and requires justification. In fact, web services are known to be rather heavy middleware in terms of resource consumption [9], which potentially jeopardizes the requirement of fresh data and minimal overhead. The main advantages of web services compared to other middleware solutions (RPC, ORB's, message queues, etc) are: (a) interoperability, meaning that they can be deployed in all platforms and configurations and (b) possibility of exporting them outside the intranet of an organization. We emphasize the interoperability property: in a large organization, there is a wide variety of data sources, involving several platforms and configurations. Web services are syntactically reliable, as they can provide a common, stable interface for the warehouse to all these sources without requiring major design and integration effort. Also, this loose coupling of sources and the warehouse results in minimal impact in the case of changes, either at the source or at the warehouse. Obviously, performance has been a concern too. Still, as we discuss in Section 4, our experiments indicate that the overall delay, incurred by the adaptation of a solution based on web services is rather small, especially if one is willing to trade resources (mainly main memory) for freshness.

In a nutshell, our contributions can be listed as follows:

- We set up the architectural framework and the issues that arise for the case of active data warehousing.
- We develop the theoretical framework for the problem, by employing queue theory for the prediction of the performance of the system. We provide a taxonomy for ETL tasks that allows treating them as black-box tasks, without the need of resorting to algebraic, white-box descriptions of their functionality. Then, standard queue theory techniques can be applied for the design of an ETL workflow.
- We provide technical solutions for the implementation of our reference architecture, achieving (a) minimal source overhead, (b) smooth evolution of the software configuration at the source side and (c) fine-tuning guidelines for the technical issues that appear.
- We validate our results through extensive experimentation. Our implementation suggests that our theoretical formulation successfully predicts the actual performance of the system.

The rest of this paper is organized as follows. In Section 2, we set up the problem theoretically and in Section 3, we present the different architectural choices and the technical challenges that each of them incurs. In Section 4, we present the experimental evaluation of the proposed framework. Finally, in Section 5, we present related work and in Section 6, we conclude with our results and present topics for future research.

2. QUEUE THEORY FOR ETL ACTIVITIES

In our architecture, data flows from the sources towards the warehouse, through an intermediate data processing stage. In this stage, data sustain various types of filtering and transformations. We employ queue theory as the cost model that predicts the data delay and the system overhead at this intermediate stage. We model each ETL activity as a queue in a queuing network. We provide a simple taxonomy for ETL activities, showing how to derive a simple queue model for them, without delving into their internal semantics. In this section, we start with some fundamentals of queue theory, then we move on to discuss a taxonomy of ETL operations and, finally, we conclude with the presentation of queue networks.

2.1. Preliminaries

Fundamentally, in a queuing model, a sequence of customers arrives at a server. If a customer arriving at the server finds the server occupied, it waits in the queue until its turn to be served comes. After the customer is served, it leaves the system [11]. If λ customers arrive at the system per time unit, then the mean inter-

arrival time is equal to $1/\lambda$. Similarly, if μ customers leave the system per time unit, then the mean service time is equal to $1/\mu$. Based on these parameters, we also define $\rho=\lambda/\mu$ as the traffic intensity which denotes the server utilization. We require that $\rho<1$ or the queue length can become unbounded.

The distribution of the arrival and the service rates can take different values (Poisson, constant, etc). Depending on these distributions, different equations hold for predicting the mean length of the queue and the mean service time for each customer. A full discussion of these properties falls outside the scope of this paper; therefore we refer the interested reader to [11, 23] for a detailed discussion.

A fundamental relation between the mean number of customers in the system N, the customer mean arrival rate in the system λ, and the mean time T that a customer remains in the system is given by Little's law. This relation is formulated as $N=\lambda*T$ and its importance resides in the fact that this equation holds for every type of queuing system irrespectively of the arrival and service rate distributions. By applying Markov Theory and Little's law to a queue with Poisson arrivals and exponentially distributed processing times, (also known as M/M/1 queue), we can estimate the mean response time of the system $W = 1/(\mu—\lambda)$ and the mean queue length $L=\rho/(1-\rho)$.

2.2. A Taxonomy of ETL Activities

Each ETL queue can direct customers to more than one subsequent queue, depending on the type of operation it performs. In queue theory, the composition of queues is treated by queue networks. The computation of the interesting properties of such networks depends on the nature of the involved individual queues. The question that arises is what kind of individual queues do the ETL activities produce. One possible way to answer this question is to define an extension of the relational algebra, specifically tailored for ETL purposes and study the properties of each operator from the viewpoint of queue theory. Since this would probably produce quite complex queues, we adopt a different, black-box approach and define a taxonomy of ETL transformations, based on the relationship of their input and output. This way, we practically categorize ETL tasks in families without delving in the particularities of their internal functionality. Specifically, the taxonomy of activities consists of the following categories: (a) Filters, (b) Transformers and (c) Binary Operations.

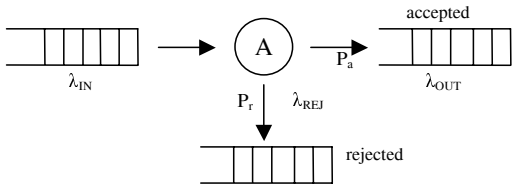

Fig. 2. Queuing model for multi-output activities

Filters examine each incoming tuple to determine whether it meets certain criteria. If these criteria are fulfilled, then a tuple is accepted and propagated towards an acceptance output. If not, it is rejected and possibly propagated towards a rejection output. We assume that tuple arrivals occur due to a Poisson process and service times follow an exponential distribution. We define the probability that some tuple i is accepted as P_a and the probability that some tuple i is rejected by the system as P_r. This is illustrated in Figure 2. It is obvious that $P_a+P_r=1$.

The filtering operations do not impose a change to the overall number of tuples making the following equation valid:

$$| \textit{tuples entering service} | = | \textit{tuples accepted} | + | \textit{tuples rejected} |$$

Also, these operations do not incur changes to the schema of the tuples entering the service facility compared to the schema of the exiting tuples. Typical operations of this category are not-null, domain and foreign key checks, selections, and in general, any type of operation, operating locally on a tuple and determining whether it will be further propagated or not. Due to their multiple outputs, filters can also act as routers for tuples whose destination depends on their value.

Considering the case of **Transformers**, tuples entering a transformer undergo changes to their value and/or their schema. We can distinguish two subclasses of Transformers taking into account the relationship between the number of tuples entering and the number of tuples exiting the transformation.

In the first case the two quantities are equal which means:

$$| \textit{tuples entering service} | = | \textit{tuples accepted} |$$

We assume that tuple arrivals occur due to a Poisson process and service times follow an exponential distribution, in other words we have the same case with filters transformations. Again, we define the probability that some tuple i is accepted as P_a and the probability that some tuple i is rejected by the system as P_r. Since all tuples are accepted, we have: $P_a=1$ and $P_r=0$ (Figure 3). Examples of such transformations are the surrogate key transformation, the usage of functions for the derivation of new values and, in general, any transformation that derives an output tuple solely on the basis of the value of a single input tuple.

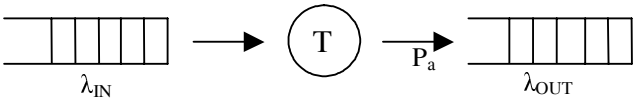

Fig. 3. Queuing model for single-output activities

In the second case, the number of tuples entering the system is different compared to the number of tuples exiting and in specific:

$$| \textit{tuples entering service} | > | \textit{tuples accepted} |$$

This occurs because some of the tuples entering service are aggregated or merged. We assume that tuple arrivals occur due to a Poisson process and service times follow an exponential distribution. The problem with this kind of transformations is that practically queue customers disappear and new customers are produced by each transformation. To model this property in terms of queue theory, we make the assumption that depending on the aggregation or merging factor, some of the incoming customers continue and some exit the system. In other words, we assume that some of the tuples, after being transformed, continue through the system as accepted. The number of these tuples equals the number of tuples produced as a result of the transformation. The rest of the tuples are assumed to be rejected by the system after their service and exit the system. The following equation holds:

$$| \textit{tuples rejected} | = | \textit{tuples entering service} | - | \textit{tuples accepted} |$$

Again, we define as P_a the probability that some tuple i is accepted and P_r the probability that some tuple i is rejected by the system: $P_a+P_r=1$. Given the aggregation factor of an incoming set of data, we can easily compute the acceptance and rejection rate as well as the respective routing probabilities. The routing probabilities are:

$$P_a = \frac{|result\_tuples|}{|input\_tuples|} \text{ and } P_r = \frac{|input\_tuples| - |result\_tuples|}{|input\_tuples|}$$

The third class of ETL activities deals with **Binary** operators. This is the case where data from multiple sources are combined and a single outgoing stream is produced. Examples of such operations involve variants of the join operation, including the join of data from different tables, as well as difference and update detection operations among different snapshots of the same table. [14] describe a window-based hash join algorithm for continuous streams. In the context of ETL, we make the following assumptions and observations:

- One of the two inputs is considered as the *primary input flow*. Tuples of this flow are checked over filters or transformed according to the values of some other relation and ultimately, either propagated towards the warehouse or rejected.
- The second input of the operator is acting as a *regulator of the primary flow*. In other words, its values are only needed in order to determine the processing and routing of the tuples of the primary flow. For all practical purposes where active ETL functionality is needed (update detection, difference, facts joined with dimension values), a static snapshot of the regulator flow can even be assumed.
- Adopting the model of [14], both inputs arrive at the same queue – they simply undergo processing with different distributions of processing times.

In principle, a binary operator has to be dealt with as a multi-class queuing system, with one class for each flow (input or output) – see Figure 4. We refer the interested reader to [14] for such a treatment. Still, based on the aforementioned assumptions, we can avoid modeling the system as a multi-class queue, and deal only with the primary flow of the operator. In the rest of the paper, we will consider single-class queues, the tuples of which either (a) continue in the system or (b) are ultimately rejected. An interesting observation here is that no matter how many different categories of tuples enter the node for service, the output tuples can be assumed to belong in one of the two aforementioned categories.

We consider Poisson arrivals and exponential service times. As stated earlier, the two routing classes are accepted with probability P_a and rejected with probability P_r and as before $P_a+ P_r=1$. This type of operations does not impose a change to the overall number of tuples existing making the following equation valid (Figure 4):

$$|\ tuples\ entering\ service\ | = |\ tuples\ accepted\ | + |\ tuples\ rejected\ |$$

However, differently from Filters, the schema of the tuples possibly changes.

We can generalize the three aforementioned classes, through a **Generic Model**, where a node consisting of a single server serves possibly more than one classes of customers. All customers arrive according to a Poisson process and are serviced with exponential service times. The general case is depicted in Figure 4.

In the general case we can assume that tuples belonging to one of the two different classes of customers, say c_i, after their ETL transformation at the node, leave the system with probability P_{ri} and continue in the queue network with probability P_{ai}. Concerning the number of tuples in the system the following equation is still valid:

$$|\ tuples\ entering\ service\ | = |\ tuples\ accepted\ | + |\ tuples\ rejected\ |$$

Concerning the schema of the tuples before and after service, we observe that the schema changes in the general case, apart from the case of filters.

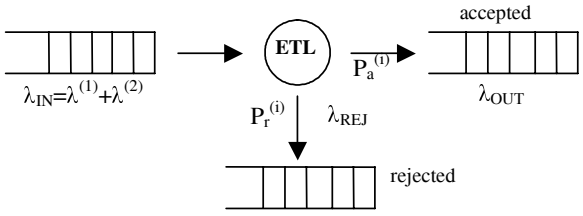

Fig. 4. Generic Model for ETL Queues

In the rest of this paper, we will follow the assumption of a primary input flow. This obviously results in forming an M/M/1 queuing node as the constructing element of our ETL queue network.

2.3. Queue Networks for ETL queues

Many queuing systems consist of a network of queues. In a *queuing network* (QN), a customer finishing service in a service facility is either immediately proceeding to another service facility or leaves the system. For our purposes, we assume that each node of this network consists of a single server with exponential arrival and exponential service times. One basic classification of queuing networks is the distinction between open and closed queuing networks. In an open network, new customers may arrive from outside (coming from a conceptually infinite population) and later on leave the system. In a closed queuing network, the number of customers is fixed and no customer enters or leaves the system. In our case, we are exclusively interested in open networks.

If an open queuing network is in steady state (i.e., the number of customers in the queue has converged over time), then for each node i, its arrival rate λ_i equals its departure rate μ_i. The arrival rate λ_i to node i is clearly the sum of all arrivals to i (including i itself). Assuming that i has N neighbors, the rate of external arrivals is λ_{0i}, and the probability of an arrival from its j-th neighbor is $p_{j,i}$, we have: $\lambda_i = \lambda_{0i} + \sum_{j=1}^{N} p_{j,i}\lambda_j$

These equations are called traffic equations and they can be transformed into a set of N simultaneous linear equations with a unique solution for M/M/1 nodes. In order to calculate the performance measures in queuing networks the steady state probabilities $\pi(k_1,\ldots, k_N)$ have to be found. The term $\pi(k_1,\ldots, k_N)$ denotes the probability of k_1 customers in queue 1, k_2 customers in queue 2 and so on. To this end, we employ Jackson's theorem that allows the calculation of the steady state probabilities of the whole network by separately calculating the probabilities of each

node, under reasonable assumptions (that our ETL queues fulfill) [23].

Jackson's Theorem [23]. If in an open network the condition $\lambda_i < \mu_i \cdot m_i$ holds for every $i \in \{1, ..,N\}$ (with m_i standing for the number of servers at node i) then the steady state probability of the network can be expressed as the product of the state probabilities of the individual nodes:

$$\pi(k_1,..., k_N) = \pi_1(k_1)\pi_2(k_2)... \pi_N(k_N)$$

Therefore, we can solve this class of networks in four steps:

1. Solve the traffic equations to find λ_i for each queuing node i.
2. Determine separately for each queuing system i its steady-state probabilities $\pi_i(k_i)$.
3. Determine the global steady-state probabilities $\pi(k_1,..., k_N)$. Derive the desired global performance measures.
4. From step 1, we can derive the mean delay and queue length for each node.

Methodology. How can we exploit the aforementioned theoretical analysis for designing ETL workflows for active data warehousing? *The design problem for active data warehousing involves predicting the mean delay and the queue length of ETL queues in the ADSA, given the source production rates and the processing power of the ADSA and the DW.*

The methodology for this task is straightforward. First, we classify each ETL task that we need to perform in one of the categories of our taxonomy. Then, we construct a queue network of such ETL queues. Finally, we solve the network equations as mentioned above.

3. FRAMEWORK AND ISSUES RAISED

Apart from the theoretical issues, there are several issues concerning the implementation of an active data warehouse. Therefore, in this section, we will start by presenting the general architecture of such a system. In subsection 3.1, we present the grand view for active warehousing and its specific instantiation that we have investigated. Then, in subsection 3.2, we proceed to a detailed presentation of the issues raised within this framework.

3.1. System Architecture

Our architecture consists of the following elements: a Data Source generating data, an intermediate data staging area that will be referred to as the Active Data Staging Area (ADSA) where the processing of data takes place and the Data Warehouse (DW). The architecture is illustrated in Figure 5.

The source comprises a data store (legacy or conventional) and an operational data management system (e.g., a DBMS or an application, respectively). Changes that take place at the source side have to be propagated towards the warehouse, which typically resides in a different host computer. The communication between hosts employs a network protocol (e.g., TCP or UDP). To avoid the extra overhead of overloading the network with half-full packets and, as our experiments indicate, to avoid overloading the source with the extra task of performing this task, we employ a *Source Flow Regulator* (SFlowR) module that compiles changes in blocks and propagates them towards the warehouse.

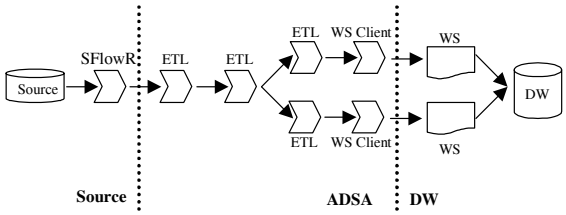

Fig. 5. Architecture Overview

Once record blocks have left the source, an ETL workflow receives them at the intermediate staging area. The role of the ETL workflow is to cleanse and transform the data in the format of the data warehouse. The ETL workflow comprises a set of ETL activities, also called *ETL queues*, each pipelining blocks of tuples to its subsequent activities, once its filtering or transformation processing is completed. In order to perform this task, each ETL activity checks its queue (e.g., in a periodic fashion) to see whether data wait to be processed. Then, it picks a specified number of records, performs the processing and forwards them to the next stage. If less than the specified records exist in the queue, then they are all retrieved. If the queue is empty, then the invocation is postponed, until there exist data to be processed.

The role of the active data staging area is versatile: (a) it performs all the necessary cleansings and transformations, (b) it relieves the source from having to perform these tasks, (c) it can act as a regulator for the data warehouse, too (in case the warehouse cannot handle the online traffic generated by the source) and (d) it can perform various tasks such as checkpointing, summary preparation, and quality of service management.

Once all ETL processing is over, data are ready to be loaded at the warehouse. As already explained, we chose to perform this task through a heavy but reliable (syntactically and operationally) middleware, web services. For each target table or materialized view at the warehouse, we define a receiving web service. To be able to invoke the web service, a client needs to be constructed. To regulate the traffic between the staging area and the warehouse, the client compiles the data in blocks, too. The web service at the warehouse side then populates the target table it serves. Load-balancing mechanisms at the warehouse side and physical warehouse maintenance (e.g., index maintenance) can also be part of this architecture. Still, for the moment, we do not address these problems.

In terms of the particular implementation that we examine in this paper, we have studied the problem as it appears over legacy sources. In our configuration, the source includes two software modules: (a) an ISAM file and (b) an application used to modify data in the legacy data source. In order to manipulate ISAM files, there is a library of ISAM routines that are invoked from the application at the source side. We have modified these library routines in order to replicate the data manipulation commands and send updates towards the staging area. Several ETL queues reside at the staging area performing cleanings, transformations and aggregations. Each ETL activity retrieves data from its queue with a constant rate, retrieving a given number of elements in constant intervals. ETL activities communicate both with each other and with the web service clients via Java thread-safe queues. The transfer from the staging area towards the Data Warehouse is done over HTTP (implying TCP as the underlying network protocol).

For our experiments, we assume that the warehouse simply stores the data performing no other task.

3.2. Issues Raised

In order to fulfill all the goals mentioned in Section 1, using the architectural elements described above, there are some issues raised which mainly concern the tuning and configuration of the system. The key issues that affect system performance and need to be resolved are discussed in this section and classified with respect to their locality at the source or the staging area, as well as the overall setup of the environment. All the technical choices and their alternatives are summarized in Table 1.

3.2.1 Choices concerning the Topology

Having described our architectural elements, the next step is to determine their topology. Our architecture offers the ability of selecting different number of tiers. Several choices exist:

- Two-tier architecture, where the source and the warehouse are found on different machines. There are two alternatives concerning this choice: the first is to place the staging area together with the source, putting the data warehouse on a separate machine. The second alternative is to place the staging area at the host where the data warehouse resides (Figure 6).
- Three-tier architecture, where we use a separate dedicated machine for the staging area, leading to a three-tier topology.

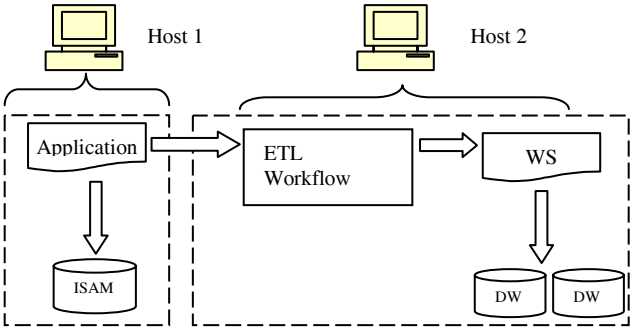

Fig. 6 Two-tier topology: The Data Warehouse and the ADSA reside on the same host, while the Source resides on a separate machine.

Coming to the two-tier architecture, the main issue that arises is related to the placement of the staging area. In the case of the staging area placed at the source, data warehousing operations do not burden the source, but still the resources used by the web services API to perform the invocation remain considerable. A way for dealing with this is to move the staging area to the warehouse host (Figure 6), which can be expected to be more powerful from the source host. This way, the source is completely detached from the active data warehousing process. Naturally, if the warehouse server is too loaded or its configuration too complex for the extra software setup of a web service server, the three-tier architecture can also be employed. Using the three tier architecture solves all the abovementioned problems, but increases the setup and maintenance cost, since an extra server, apart from the one used from the warehouse, has to be engaged and administered.

Having discussed the architectural alternatives for our topology, we can now proceed to discuss the technical issues raised for each of the main components and their overall setup.

3.2.2 Choices concerning the Source

Concerning the source side, the first consideration that arises has to do with the interconnection type between the source and the staging area. Since our goals are to impose as little impact as possible to the source and to make only minor changes, we have chosen the solution of sockets both due to its anticipated (but not thoroughly tested) lighter footprint characteristics and the easiness of programming such a solution.

The next choice is between TCP and UDP protocols for the transmission of data between the source and the staging area. On one hand, TCP offers reliability. On the other hand, UDP offers speed through non-blocking calls, followed by a concern on the server side for the socket buffer size, in case of extended datagram bursts and no reliability.

A third architectural choice concerns the way that changes to the source file are written to the socket, i.e., whether data are organized in blocks before being further propagated to the staging area. There are two ways to deal with this issue: either to write each modification to the socket, or to write bulks of modification commands. In the first case, whenever a data manipulation command is issued, it is immediately written to the socket along with the respective data. In the second case, nothing is written, until a number of records is completed. Then, all records together are sent to the staging area.

3.2.3 Choices concerning the Staging Area

The internal structure of the data staging area and the tuning of its operation are the major issues concerning the performance of our architecture. The staging area is a multithreaded environment with shared components, thus having to be set up properly to avoid race conditions and consistency.

The problem of locking raises the issue of queue emptying rate. Assuming that the input to the staging area is determined by the workload of the source (i.e., it cannot be constrained by the warehouse administrator), a proper emptying rate for the ETL queues has to be determined. A high arrival rate compared to the configured service rate will result in instability and queue length explosion. On the contrary, a very high service rate potentially results in too many locks of the queue (resulting again in delay, contrary to what would normally be expected). It is obvious that the service rate should be close to the arrival rate in order to have both efficient service times, and as less locks as possible.

Another dilemma is related to the interconnection type between the staging area and the data warehouse. As already mentioned, the staging area invokes a web service residing at the warehouse side. Although the SOAP protocol is one-way and asynchronous, implementations abide by the traditional middleware conventions of remote invocation, namely (a) *blocking* and (b) *non-blocking*. Blocking invocation involves an acknowledgment message to be sent from the web service, before its client can continue. In our case, this means that a response from the warehouse is required, delaying however the queue emptying rate. Non-blocking invocation does not delay the queue-emptying process of the web service client, since no response is returned from the invocation.

Finally, the issue of sending data as tuple-at-a-time or blocks is raised again for the communication between the staging area and the warehouse. In this case, apart from the network overhead, the cost of parsing the incoming web service messages at the warehouse plays a role for this choice.

3.2.4 Choices concerning the Warehouse

The data warehouse side is characterized by a web wervice per target table, receiving the cleansed data from the data staging area. The web services API offers three ways of handling the remote invocations of the client that resides in the data staging area. The first way is to create a single web service instance that handles all incoming requests. The second way is to create an instance for every session, and the third is to create an instance for each invocation request. In our configurations, we use the first of these alternatives. The reason is that in our experiments, we have employed one client for the service, which stops its operation after inserting a specific amount of records into the ISAM file. This makes the case of using an instance per session the same as using a single instance. Using an object per request is prohibitive, since we assume high frequency invocations.

Table 1. Architectural choices

Issue	Alternatives
General Architecture	
Topology	- 2-tier, ADSA at the source side - 2-tier, ADSA at the DW side - 3 tier
Source	
Connection Type	- UDP - TCP
Propagation Type	- One at a time - Block-based
Active Data Staging Area	
Interface between the two APIs	- None - Synchronized Queue
Web Service invocation type	- Blocking - Non Blocking
Propagation Type	- One at a time - Block-based
Data Warehouse	
Session management	- Single WS - Instance per session - Instance per request

4. EXPERIMENTS

In this section, we present the experiments we conducted. We present two sets of experiments. The first set presented in section 4.1 deals with the general behavior of the system. The purpose of this set of experiments is to figure out the behavior of each system component separately, and to establish guidelines for building the system. In this case, data are just transferred to the warehouse and no ETL operations are involved. In the second set of experiments, presented in section 4.2, we evaluate the behavior of our system in a realistic setup, based on the conclusions derived from the first set. Naturally, in this case, we also transform data using ETL operations.

Our experimental setup, which stands for both cases, is as follows: The ISAM library that we altered is the PBL/ISAM suite [20] available under GPL license. We have used a sample program distributed within the suite as the legacy application. We use two different data sets for our purposes. The first consists of 100,000 records and the second of 1,000,000 records. The ETL queues of the ADSA have been implemented using the Sun JDK 1.4, whose runtime engine has also been used. As a Web Services platform we have used Apache Axis 1.1 [4] with Xerces XML parser running over Apache Tomcat 1.3.29. Our data warehouse is implemented as a MySQL 4.1 database.

The host we used for the source was a PIII 700MHz with 256MB of physical memory running SuSE Linux 8.1. The host used as the data warehouse was a Pentium 4 2.8GHz with 1GB of physical memory running Mandrake Linux. This server also hosted the staging area. The hosts are interconnected via the switched Fast Ethernet LAN of our department.

Our data were created from the TPC-H data generation tool. For the first case, each row of data has fixed size equal to 20 bytes. In the second case, where we evaluate the system behavior under operational conditions, we used data of variable size. In this case each row has an average size of 140 bytes.

In our experiments we evaluate the cost in marginal conditions. Thus in order to evaluate the worst case, the source stores data at its peak capability. Moreover, since our warehouse host is a much faster computer than the source host, we would not be able to make safe conclusions if we let it operate at full capability (see also subsection 4.1.4). Thus we simulate slower server performance by employing timeouts between operations. This will be explained in more detail later.

4.1. Experiments on Architecture without ETL Processing

This section includes the first set of experiments we conducted. The aim of these experiments is to decide on basic architectural choices of our system. Throughout the experiments, the software operating at the staging area is a simple queue, called *Data Warehouse Flow Regulator* (DWFlowR), receiving source blocks of records and passing them to the warehouse.

4.1.1 Smooth Upgrade

One of the goals of our architecture is to pose minimal modifications to the source's code. In our approach, we do not alter the legacy application itself, but the library that manipulates the ISAM files by adding few lines of code to the routines that are of interest to the purpose of active warehousing. These routines are: the file opening routine, the record insertion routine and the file closing routine. The alterations are located only in the following four points of the library's source code:

1. The first modification is to include our library which contains the socket's client and the SFlowR.
2. The second modification is to add a call to the routine of our library that opens a socket to the staging area at the ISAM file opening routine. This call is performed only if the opening of the ISAM file is successful.
3. The third modification is to extend the insertion routine of the ISAM file library that writes the record to the file with a call to our library's function that propagates the change to the socket. This routine stores the specific record to the

SFlowR's buffer and when the defined number of records is completed, it delivers them to the staging area. Again, this routine is called only after a successful insertion.

4. The fourth modification is to add a call to the routine of our library that closes the opened socket to the staging area, at the ISAM file closing routine. This call is performed only if the closing of the ISAM file is successful.

Figure 7 shows the alterations that we have performed to the library in pseudo-code. The overall length of code that had to be written for this part of the implementation, including the additions at the ISAM library, is approximately 100 lines.

The routine that opens the socket to the DWFlowR reads configuration information from a plain text file, before the opening of the socket. This file contains the following three pieces of information:

1. The number of records the SFlowR will gather
2. The address of the DWFlowR
3. The port of the DWFlowR

Original Routine	Altered Routine

`Open_isam_File()`
`{`
`  …`
`opening_isam_file_commands`
`  …`
`}`
	`Open_isam_File()`
	`{`
	`  …`
	`opening_isam_file_commands`
	`  …`
	`if(open==success)`
	`DWFlowR_socket_open()`
	`}`

`Write_record_to_File()`
`{`
`  …`
`insert_record_commands`
`  …`
`}`
	`Write_record_to_File()`
	`{`
	`  …`
	`insert_record_commands`
	`  …`
	`if(write==success)`
	`write_to_SFlowR()`
	`}`

`Close_isam_File()`
`{`
`  …`
`closing_isam_file_commands`
`  …`
`}`
	`Close_isam_File()`
	`{`
	`  …`
	`closing_isam_file_commands`
	`  …`
	`if(close==success)`
	`DWFlowR_socket_close()`
	`}`

Fig. 7. Code alterations at the routine opening the ISAM file.

As an overall assessment of the impact of our changes, we can say that (a) minimal code had to be written to achieve the replication of incoming updates to the warehouse in an active fashion, (b) simple configuration parameters are required, (c) no changes were required to the code, rather than a simple recompilation under the new library.

4.1.2 UDP vs. TCP

The first parameter that needed to be tested involved the network protocol between the source and the staging area. The goal of our first experiment is to determine the system's behavior using UDP and specifically if there are any datagram losses. The results show a 35% packet loss of data, most probably due to the overflowing of data. Such losses are prohibitive for normal operation of an on-line environment. Therefore, for the rest of the paper, we have fixed TCP as the interconnection protocol between the source and the staging area.

4.1.3 Overhead at the Source

The main requirement for the architecture at the source side involves minimal overhead during regular operation. Therefore, the goal of the next experiment is to measure the overhead that our configuration incurs at the source side. We measure the time to complete the insertion of (a) 100 000 and (b) 1 000 000 to the ISAM file.

First, we measure the effect of using the SFlowR at the source. We try three values: 1, 100, and 1000 records for each packet that the SFlowR sends to the staging area. When using one record at a package, we have in fact the case of not using a SFlowR. In Fig. 8 and 9, we refer to the regular operation of the source (without sending records towards the ADSA) as "plain".

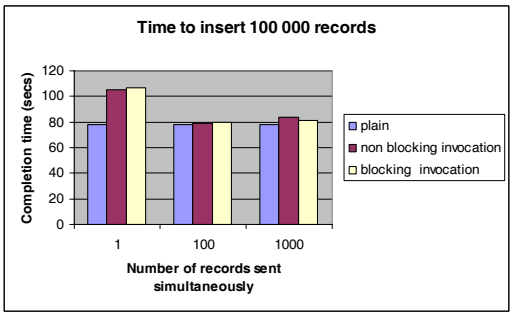

Fig. 8. Time to insert 100 000 records using two-tier topology

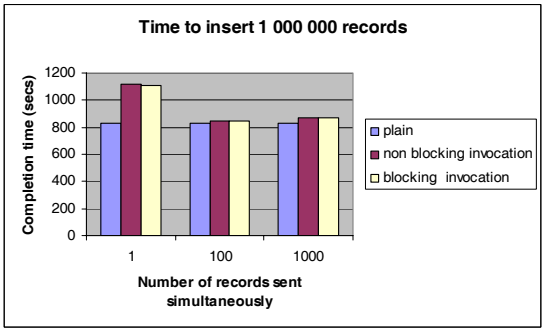

Fig. 9. Time to insert 1 000 000 records using two-tier topology

Another issue worth investigating is the isolation of the Source, ADSA and Data Warehouse layers. Therefore, we employ two modes for the operation of the staging area, to assess its impact. Each test case is examined with blocking and non-blocking invocation for the communication between the staging area and the Web Service at the data warehouse side. The staging area uses a synchronized queue. The input rate at the queue is equal to the output rate of the Legacy Application. The queue's output rate is fixed to one thousand records per second.

Figure 8 depicts the results of the experiment for 100 000 records, while Figure 9 the results for 1 000 000 records. The x-axis for Figures 8 and 9 shows the number of rows in a packet. The y-axis of the diagrams measures the throughput of inserting the records to the ISAM file.

Based on our experimental results, the following observations are made:

1. The SFlowR plays a very important role, since without it the throughput deteriorates by 34%, while using a SFlowR incurs an impact of approximately 1.7%.
2. The way that the DWFlowR is tuned does not affect the source. Regardless of using blocking or non blocking Web Service invocation at the DWFlowR, the source's throughput is the same in both cases.
3. Sending smaller packets of records performs slightly better, since in the case of 1000 records, network propagation time decreases throughput. Moreover, choosing a packet size of 100 instead of 1000 records saves buffer size at the SFlowR.
4. The cost delay ratio in terms of the size of data sent to the warehouse remains stable both in the case of 100 000 and 1 000 000 records.
5. The behavior of our system remains stable regardless of the size of data it has to handle.

4.1.4 Data Freshness

A major requirement in our setting is to achieve the maximum data freshness possible, through our framework. With a 1.7% delay at the source, the focus of interest is isolated in the side of the staging area. The goal of the next set of experiments is to measure the data freshness time provided by our application with respect to the queue emptying rate and the block retrieved from the queue. We consider as *data freshness time* the time required for a record that was inserted in the ISAM file to be transferred to the warehouse.

Specifically, we measure the overall throughput, i.e., the time needed to empty the DWFlowR's queue after the first record is sent to the warehouse. The freshness is then measured as the time needed to empty the queue, which practically stands for the response time for the last record. To perform these measurements, we assume that the legacy application sends 100 000 records to the staging area in blocks of 100 records over TCP. Also, we measure the queue length as an indicator of resource consumption at the staging area.

It is important to determine the behavior of the ADSA using data service rates close to the service rate of the source. Since our data warehouse server is faster than our source, we wanted to simulate slower performance to determine the behavior of the system in marginal conditions. Thus, we empty the queue retrieving the records from the queue using timeouts of 0.1 seconds and retrieving 100, 150 and 200 records each time and then invoking the web service, having as a source data rate approximately 1300 records per second. These are the maximum emptying rates, meaning that if the queue contains fewer records, then all the records from the queue are retrieved. We also present the results of the server operating at its top performance.

The results of emptying the queue using various rates are depicted in Figure 10. In these graphs, two other parameters play a major role. The first parameter, as indicated on the x-axis, is the time required to empty the queue. The second parameter, as shown on the y-axis, is the number of elements in the DWFlowR's queue. Figure 11 depicts the data freshness provided by our architecture.

We measure the time required to transfer all data from the staging area to the data warehouse.

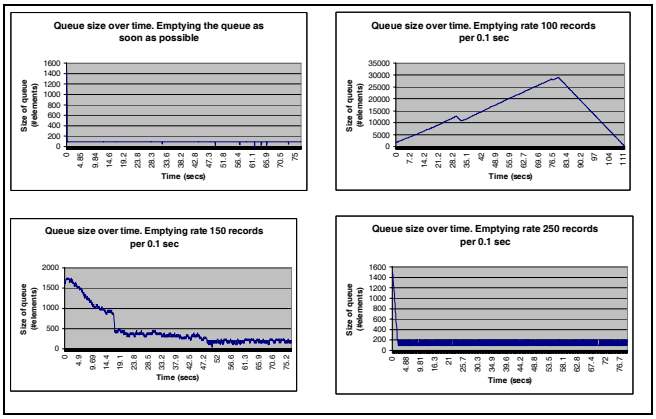

Fig 10 Queue size at the staging area emptying the queue as soon as possible

In Figure 10, the top left graph shows what happens when we let the ADSA operate fully. We can easily see that practically no queue is ever formed. The mean queue size is 100 records which is the rate of the SFlowR. In other words, the ADSA is one step later than the source, in terms of performance.

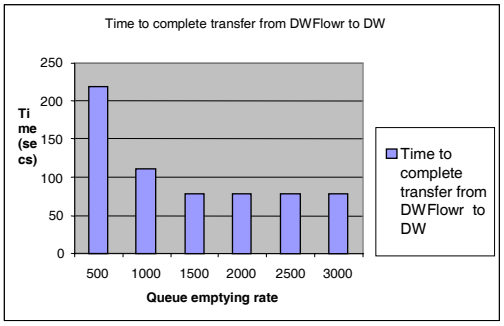

Fig. 11 Queue emptying time at the staging area.

The other three graphs show the queue sizes using service rates of 1000, 1500 and 2000 rows per second. In the first case, where the service rate is lower than the arrival rate, the queue explodes, as expected. In the second case, where we are close to the arrival rate, the queue displays a quite transient yet stable behavior. The last graph practically presents the same behavior as in the first graph even though the service rate is slightly increased compared to the case of 1500 rows per second. We have also experimented with even higher service rates i.e., up to 3000 rows per second, which still present the same behavior. We omit these results due to lack of space.

Observing the results of this set of experiments, we are led to the following conclusions:

1. We can achieve data freshness time equal to data insertion time when we *continuously* empty *a small size* queue.
2. In this case, the size of the queue is equal to the arrival rate from the source, i.e., there is practically no delay at the queue.

4.2. Operational Evaluation

In this subsection, we will use the architectural guidelines derived from the first set of experiments presented in subsection 4.1 to build an active data warehouse where we will also deploy our online ETL operations. The aim of this section is to evaluate the behavior of this fully deployed system.

4.2.1 Impact at the Source

In this paragraph, we will try to refine the results learned in 4.1. For this reason, we examine again the impact on the source system of the packet size of the SflowR. This time we will use small package sizes, as derived from the previous set of experiments.

Figure 12 shows the impact at the source using packets at the SflowR of various sizes. In general, packet sizes of over 25 records offer the least burden to the source. The smallest delay was achieved with a packet size equal to 50, where the source delay was measured to be at 5.8%.

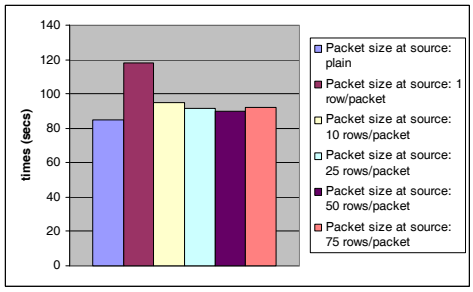

Fig. 12 Packet size of the SFlowR and impact at source

4.2.2 Data Freshness of Online ETL

In this paragraph, we deploy certain ETL scenarios and evaluate their performance compared to the theoretical analysis and in terms of data freshness. For this reason, we consider the following scenarios and their individual steps:

- Scenario (a): We simply transfer data inserted into the legacy application to the warehouse using various service rates.
- Scenario (b): (1) We filter 10% of incoming data through a selection predicate. (2) Then, we employ a surrogate key transformation to the first column of the filtered data. (3) Next, we perform a cumulative aggregation (group by with sum). (4) Finally, data are fed to the warehouse.
- Scenario (c): (1) We filter 10% of incoming data. (2) Then, we additionally filter another 2% of the remaining data. (3) Next, a surrogate key operation is applied to the first column of the data. Then, the stream is replicated along two branches.
    - For the first branch populating a materialized view, (4.1.1) a cumulative aggregation is performed and (4.1.2) data are fed to the warehouse.
    - For the second branch, populating the detailed fact table (4.2), data are fed to the warehouse.
- Scenario (d): (1) We filter 10% of incoming data. (2) We replace the values of the first field, to simulate value computations through functions. (3) A surrogate key transformation is applied. Then, the stream is replicated along two branches:
    - For the first branch, (4.1.1) a cumulative aggregation is performed first and (4.1.2) a filter (HAVING clause) rejecting 6% of the groups is applied. Then, (4.1.3) data are fed to the warehouse.
    - For the second branch, (4.2.1) a second value derivation is performed, (4.2.2) a filter rejecting 2% of detailed input data is applied and, finally, (4.2.3) data are fed to the warehouse.

In Figures 13, 14, 15 and 16 we depict the evolution of the experiments as time passes. The x-axis depicts the time points when we measured the queue length. The final time point gives the time (in seconds) required to complete the transfer from the ADSA to the Warehouse. The y-axis depicts the number of rows existing in the queue. The graphs only show the time points when our measurement showed that the queue is not empty. Each of the queues in the graph is identified by its operation name (e.g., in Figure 13, "FILTER"), possibly its selectivity (e.g., "10" for 10%) and its occurrence in the scenario (e.g., "01" for the first occurrence).

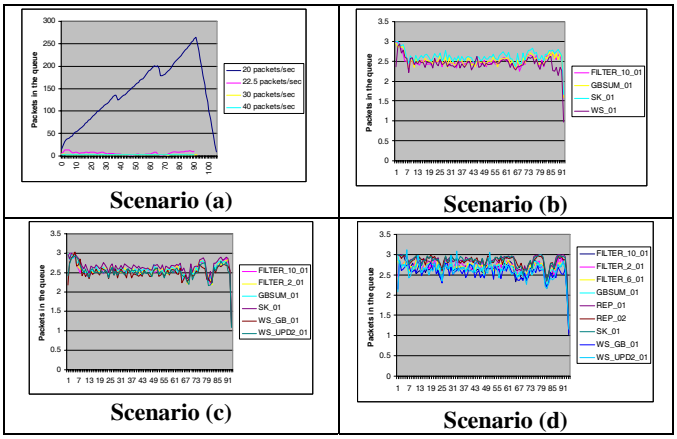

Fig. 13-16 Queues for scenarios (a), (b), (c), (d)

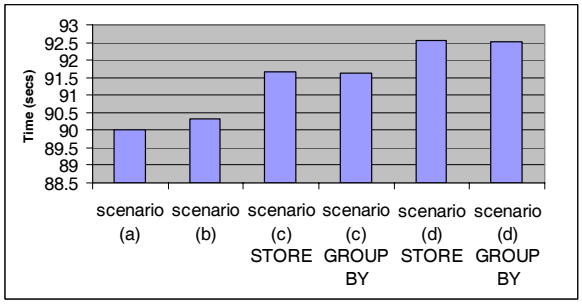

Fig. 17 Data freshness for each scenario

In all the scenarios the block size of the SFlowR was fixed at 50 rows per block. Scenario (a) was configured to use the following service rates: 20, 22.5, 30 and 40 packets per second, which represent rates of 1000, 1250, 1500 and 2000 rows per second respectively. In scenarios (b), (c) and (d) the service rates were simulated to 30 packets per second both for the ETL rates and the Web Service clients.

Finally, Figure17 summarizes the total times needed for the ADSA to transfer all data to the warehouse, for each scenario of ETL queues.

Observing the figures, we derive the following conclusions:

1. The source capability is approximately 1100 rows/sec. In scenario (a) we are led to queue explosion, when we employ service rate smaller than the source's arrival rate. Using a service rate of 1250 rows / sec, which is a setting close to the arrival rate, we can see that transient effects tend to appear, but the queue converges to steady state. By using higher service rates, 1500 and 2000 rows / sec respectively, the queue maintains its steady state.
2. In scenarios (b), (c) and (d) we observe that the entire system, as well as the queue of each operation, maintains a steady state. The number of packets in the queue is less or equal to the maximum number of packets polled simultaneously from the queue. This practically means that after each poll the queue empties and that the ADSA is only one step behind the source.
3. In Figure 17, the total time needed for the entire dataset to be transferred from the ADSA to the Warehouse is dependent on the number of the intermediate ETL operations. As the number of intermediate ETL operations that a packet has to visit increases, the total delay increases as well. Nevertheless, in our exemplary scenarios, the increase is rather small, due to the pipelining of data. The average delay per row is around 0.9 msec for all scenarios.

In Table 2 we present the comparison of our theoretical evaluation of queue length against the observed values. For lack of space, we show only the results of scenario (c) with service rate of 2000 rows/sec; all the other scenarios present identical behavior. As one can observe, in average, the theoretical prediction typically underestimates the average queue length by a very small amount (of the size of 5 records). In our detailed experiments, the system behaves in accordance with this pattern for all four scenarios, with an average error of half a packet (i.e., 25 records).

Table 2. Theoretical prediction vs. actual measurements of average queue length for scenario (c) in packets

	Measured	Theoretical Prediction	Difference
FILTER_10_01	0.160	0.056	0.104
FILTER_02_01	0.134	0.047	0.087
SK_01	0.154	0.054	0.100
GB_SUM_01	0.137	0.048	0.089
WS_GB	0.091	0.031	0.059
WS_GB_UPD	0.100	0.035	0.066

5. RELATED WORK

In this section, we present work related to our approach. Research in ETL has provided results in (a) tools [10, 21], (b) algorithms for specific tasks [7, 15, 16, 18]. Both tools and algorithms operate in a batch, off-line fashion. So far, minimum emphasis has been paid to the investigation of ETL tasks, apart from a general model for [7, 18], where ETL activities are studied under the prism of lineage or resumption of a failed process. As already mentioned, data streams [1, 5, 17] could possibly appear as the paradigm for active warehouse maintenance. So far, streams have been studied from the point of view of continuous querying, without any investigation of transformations or updates. Both our architecture and theoretical analysis could possibly be applied over streams for this purpose. To our knowledge, the only paper related to our approach is [14], where the authors apply a "white-box" (as opposed to our black box) method to determine the properties of SPJ relational operators with respect to queue theory.

Work in materialized views refreshment [12, 13, 24, 25] is orthogonal to our setting. In [13] the authors describe materialized views, their applications, and the problems and techniques for their maintenance. Novel techniques and an up-to-date survey of related work in the field are presented in [12]. Materialized views refreshment fits orthogonally with our on-line refreshment technique, since we can treat each ETL queue as a black-box process. In the context of this paper, a dedicated web service is assigned to each materialized view. Although the tuning of the system for large workloads of views is an interesting topic of research, we find this issue outside the scope of this paper.

Another area related to our approach is the one of active databases. In particular, if conventional systems (rather than legacy ones) are employed, one might argue that the usage of triggers [7] could facilitate the on-line population of the warehouse. Still, related material suggests that triggers are not quite suitable for our purpose, since they can (a) slow down the source system and (b) require changes to the database configuration [6]. In [19] it is also stated that capture mechanisms at the data layer such as triggers have either a prohibitively large performance impact on the operational system. As compared to these problems, our architecture achieves low overhead with minimal impact in the configuration of the source. We conjecture that a replication mechanism similar with the proposed one, propagating log entries towards the warehouse is a possible solution towards this problem.

6. CONCLUSIONS AND FUTURE WORK

Active Data Warehousing refers to a new trend where data warehouses are updated as frequently as possible, due to the high demands of users for fresh data. In this paper, we have proposed a framework for the implementation of active data warehousing, keeping in mind the following goals: (a) minimal changes in the software configuration of the source, (b) minimal overhead for the source due to the "active" nature of data propagation, (c) the possibility of smoothly regulating the overall configuration of the environment in a principled way. In our framework, we have implemented ETL activities over queue networks and employed queue theory for the prediction of the performance and the tuning of the operation of the overall refreshment process. In terms of data freshness, source overhead and minimal impact of software configuration the results seem satisfactory. A summary of the lessons learned is as follows:

- In terms of architecture, isolating the ETL tasks in a special-purpose area, either in the warehouse, or in an intermediate tier, guarantees both minimum performance overhead at the source and the possibility of regulating the flow towards the warehouse target tables.
- Queue theory can be successfully employed as the theoretical background for the estimation of the response of the active staging area. The system reaches a steady state quite close to the predicted behavior. Freshness is quite satisfactory too.

- The overall overhead at the source side is around 1.7% and the amount of code modification is around 100 lines, without affecting applications.
- Tuning the network-related parameters helps. TCP should be used instead of UDP, due to the packet loss of the latter. Organization of rows in blocks, both at the source and the ADSA side increases performance.

Future work includes several directions. A first line of research would have to do with the failure management of the components of the environment, to determine safeguarding techniques and fast resumption algorithms for the event of a failure. Further tuning can be made, by testing multiple concurrent loading sources for the warehouse. Also, the case of materialized aggregate views and schema evolution poses interesting challenges in this context.

7. ACKNOWLEDGMENTS

E. Papapetrou has helped with comments on issues of queue theory and implementation. This research has been partially supported from the European Commission and the Greek Ministry of Education through the Pythagoras Program.

8. REFERENCES

[1] Daniel J. Abadi, Don Carney, Ugur Çetintemel, et al. Aurora: a new model and architecture for data stream management. *The VLDB Journal, 12(2)*, 120-139, 2003.

[2] G. Alonso, F. Casati, H. Kuno, V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2003.

[3] J. Adzic, V. Fiore. Data Warehouse Population Platform. In *Proc. 5th Intl. Workshop on the Design and Management of Data Warehouses (DMDW'03)*, Berlin, Germany, 2003.

[4] Apache Software Foundation. Axis. Available at http://ws.apache.org/axis/

[5] S. Babu, J. Widom. Continuous Queries over Data Streams. *SIGMOD Record 30(3)*, 109-120, 2001.

[6] Donald Burleson. New Developments In Oracle Data Warehousing. Available at: http://dba-oracle.com/oracle_news/2004_4_22_burleson.htm

[7] Stefano Ceri, Jennifer Widom. Deriving Production Rules for Incremental View Maintenance. In *Proc. VLDB, Barcelona Spain, September 1991*, 577-589

[7] Yingwei Cui, Jennifer Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal 12(1)*, 41-58, 2003.

[9] W. Duquaine Web Services Ruminations. Presentation at *High Performance Transaction Systems Workshop (HPTS'03)*. Asilomar Conference Center, California, October 12-15, 2003. Available at http://research.sun.com/hpts2003/

[10] Galhardas, H., Florescu, D., Shasha, D., and Simon, E.. Ajax: An Extensible Data Cleaning Tool. *In Proc. ACM SIGMOD*, Dallas, Texas, May 2000, p. 590.

[11] D. Gross, C. Harris. *Fundamentals of Queuing Theory*. Wiley, 3rd Edition, 1998.

[12] H. Gupta and I.S. Mumick. Incremental Maintenance of Aggregate and Outerjoin Expressions. To appear in Information Systems, 2004.

[13] Ashish Gupta, Inderpal Singh Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *Data Engineering Bulletin 18(2)*, 3-18, 1995.

[14] Qingchun Jiang, Sharma Chakravarthy. Queueing analysis of relational operators for continuous data streams. In *Proc. CIKM*, New Orleans, Louisiana, USA, November 2003, 271-278.

[15] Wilburt Labio, Jun Yang, Yingwei Cui, Hector Garcia-Molina, Jennifer Widom: Performance Issues in Incremental Warehouse Maintenance. In *Proc. VLDB*, Cairo, Egypt, September 2000, 461-472.

[16] Wilburt Labio, Hector Garcia-Molina: Efficient Snapshot Differential Algorithms for Data Warehousing. In *Proc. VLDB*, Mumbai, India, September 1996, 63-74.

[17] D. Lomet, J. Gehrke. Special Issue on Data Stream Processing. *Data Engineering Bulletin*, *26(1)*, 2003.

[18] Wilburt Labio, Janet L. Wiener, Hector Garcia-Molina, Vlad Gorelik. Efficient Resumption of Interrupted Warehouse Loads. In *Proc. of ACM SIGMOD*, Dallas, Texas, USA, May 2000, 46-57.

[19] On-Time Data Warehousing with Oracle10g - Information at the Speed of your Business. An Oracle White Paper. August 2003. Available at http://www.oracle.com/technology/products/bi/pdf/10gr1_twp_bi_ontime_etl.pdf

[20] P. Graf. The Program Base Library. Publicly available through http://mission.base.com/peter/source/

[21] Vijayshankar Raman, Joseph M. Hellerstein: Potter's Wheel. An Interactive Data Cleaning System. In *Proc. VLDB*, Rome, Italy, September 2001, 381-390.

[22] C. White. Intelligent Business Strategies: Real-Time Data Warehousing Heats Up. *DM Peview*, August 2002. Available at http://www.dmreview.com/article_sub.cfm?articleId=5570

[23] A. Willig. Performance Evaluation Techniques. Available at http://www-ks.hpi.uni-potsdam.de/docs/engl/teaching/pet/ss2004/script.pdf, 2004.

[24] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, Jennifer Widom: View Maintenance in a Warehousing Environment. In *Proc. of ACM SIGMOD*, 1995, 316-327.

[25] Xin Zhang, Elke A. Rundensteiner: Integrating the maintenance and synchronization of data warehouses using a cooperative framework. *Information Systems 27(4)*, 219-243, 2002.