Data Integration: Inconsistency Detection and Resolution Based on Source Properties

Philipp Anokhin, Amihai Motro

George Mason University, Fairfax, VA, USA

Abstract. This paper addresses the problem of integration of multiple heterogeneous information sources. The sources may conflict with each other on the following three levels: their *schema*, *data representation*, or *data themselves*. Most of the approaches in this area of research resolve inconsistencies among different schemas and data representations, and ignore the possibility of data-level conflict altogether. The few that do acknowledge its existence are mostly probabilistic approaches which just detect the conflict and provide a user with some additional information on the nature of the inconsistency (e.g. give a set of conflicting values with attached probabilities). We propose an extension to the relational data model that makes use of meta-data of the information sources called *properties*. This extension gives ground to a flexible data integration technique described in this work. The process of data integration for a particular user query consists of construction of the data blocks such that their extended union constitutes the query result, data conflict detection and data conflict resolution. An improvement to data clustering techniques in the conflict detection phase is also presented in the paper. It uses another type of meta-information available from the sources (source descriptions in terms of the virtual database schema) to narrow down the areas of possible data conflicts. For the conflict resolution phase, a flexible algorithm is offered. The algorithm is guided by user-defined importance (or *weights*) of properties and by expert-defined resolution strategies that incorporate the domain knowledge.

1 Introduction

This paper addresses the data integration problem: there exist multiple heterogeneous autonomous information sources that need to be integrated; users want to be able to query the integrated sources transparently and to receive a single, unambiguous answer.

An important issue in the data integration problem is a possibility of *conflict* among the information sources. The sources may conflict with each other on different levels:

- **Schema** level, i.e. different data models or different schemas within the same data model.
- **Data representation** level, e.g. data in different natural languages, measurement systems, etc.
- **Data value** level, i.e. factual discrepancies exist among several sources in the values for the same objects.

Note that each level of conflict can only be observed when the previous level is resolved. That is, different attribute names in different information source schemas have to be mapped to each other before discrepancies in measurement systems can be detected. Similarly, different attribute values have to be within the same measurement system to infer that these values indeed contradict each other.

The resolution of schematic inconsistencies is often performed in the process of interpreting a user query. Numerous approaches exist that attempt to enumerate all possible *translations* of the query into a query over the source descriptions [11], [2], [7], [3]. The optimal translation (either by the retrieval cost or the number of the participating sources) is chosen as an answer to the original query.

A good number of approaches also resolve inconsistencies at the data representation level by providing means for aggregation and conversion of values to a uniform standard (e.g. [17], [11]).

The aforementioned information integration systems provide their users with different data integration techniques. Yet, with exception of HERMES ([18]) and Multiplex ([13]), none of them deals with the issue of possible data inconsistencies in the information sources that are to be integrated. However, HERMES does not attempt to detect data value conflicts, instead it relies on the creator of the mediator to manually specify a conflict resolution method when a conflict might occur. Also the available resolution methods are limited to a number of predefined fixed strategies. The Multiplex system both detects data inconsistencies and attempts to resolve them. However, it has several disadvantages: (1) Instead of a single answer users are provided with its consistent and inconsistent parts (possibly with multiple levels of inconsistency). (2) No additional information on the sources is used — they all are assumed to be "equal". (3) Users have no control over the inconsistency resolution process. (4) The Multiplex model defines data value-level conflict as existence of two or more candidate answers to the same query that are not equal. Consequently, two candidate answers such that one is a subset of another are considered inconsistent, although there is no factual discrepancy between them, but rather discrepancies on the *completeness* of the answers.

In the area of logic databases, [4] research possibilities of obtaining consistent answers from inconsistent databases (with data-value level inconsistencies as a special case). They define "repaired" consistent database instances, but concentrate on obtaining query answers consistent in all possible repaired instances, rather then preferring one repair over the other. The repairs are also limited to adding/deleting tuples to the original DB instance.

A few approaches exist that resolve the data value-level conflicts based on the content of the conflicting data and possibly some probabilistic information that is assumed to be available. They only detect the existence of the data inconsistencies and provide their users with some additional information on their nature (e.g. [1], [8]) or try to resolve the data-level conflicts through use of probabilistic data by returning a probabilistic value called *partial value*: a set of conflicting values with attached probabilities [6], [19], [12]. However, some important problems have not been addressed by the aforementioned probabilistic works: (1) Probabilistic information has to be provided for every datum in an information source (which is relatively rare, especially for web-based sources). (2) Probability-based inconsistency resolution is only partial: it does not offer a single answer, only a set of values with their probabilities. (3) Many of the probabilistic approaches to the data value-level conflict resolution attempt to combine probabilistic data (i.e. meta-data) rather then data: compute probabilities of a predefined set of data from the conflicting probability values, rather then compute data themselves from the conflicting data values. (4) Even when two different partial values are inconsistent, it is often assumed that their intersection is not empty. Otherwise, the conflict cannot be resolved (5) Users can not influence the resolution process to arrive at the "best in quality" answer.

With the growth of the Internet, a large number of data sources emerged that compete with each other trying to win a larger customer base. Increasingly, these data sources provide potential users with information about the sources (i.e. *meta-information*), that would allow users to make judgment about the suitability of a particular information source for the intended use. These meta-information is referred to as source *properties*. Examples of properties include (but are not limited to):

– **Timestamp**: time when the data in the source was validated.
– **Cost**: network retrieval cost and/or amount of money to be paid for the information.
– **Clearance**: clearance level needed to access the source data.
– **Priority**: authoritative source ranking supplied by the user or a group of experts.

We propose a solution for data value-level conflicts through the use of such source properties. The data model (the relational data model was chosen as a framework for the data integration algorithm) and the notion of a database query are extended to include properties. This extension gives ground to a flexible data conflict detection and resolution technique described in this paper.

2 Background

2.1 Schemas and Instances

Let $\mathcal{A}$ be a finite set of attributes, where each attribute $A \in \mathcal{A}$ has a finite domain $dom(A)$. Assume a special value *null* which is not in any of the domains.

– A *relation schema* R is a sequence of attributes from $\mathcal{A}$. A *tuple* t of a relation schema $R = (A_1, \ldots, A_m)$ is an element of $dom(A_1) \cup \{null\} \times \cdots \times dom(A_m) \cup \{null\}$. A *relation instance* r of R is a finite set of tuples of R.
– A *database schema* D is a set of relation schemas $\{R_1, \ldots, R_n\}$. A *database instance* d of D is a set of relation instances $\{r_1, \ldots, r_n\}$, where r_i is a relation instance on the relation schema R_i.
– Finally, a *database* is a pair (D, d), where D is a database schema and d is a database instance of the schema D.

2.2 Views and Queries

Let D be a database schema. A *view* of D is a relational algebra expression that defines (1) a new relation schema V called *view schema*, (2) for any instance d of D — an instance v of V called the *extension* of V in the database instance d.

A *query* Q on a database schema D is a view of D. An extension of Q in a database instance d of schema D is called an *answer* to Q in the database instance d.

2.3 Schema Mappings and Multidatabases

Consider a database (D, d). Let D' be a database schema whose relation schemas are defined as views of the relation schemas of D. Let d' be the database instance of D' which is the extension of the views D' in the database instance d. A view V of D and a view V' of D' are *equivalent*, if for every instance d of D the extension of V in d and the extension of V' in d' are identical. Intuitively, view equivalence allows to substitute the answer to one query for an answer to another query, although these are different queries on different schemas.

Assume database schemas $D_1, \ldots, D_n$, which are defined as views of the relation schemas of D. A *schema mapping* m is a collection of view pairs (V, V_i), such that V is a view of D, V_i is a view of one of $D_1, \ldots, D_n$, and V is equivalent to V_i.

Then a *multidatabase* consists of (1) a *global* schema D, (2) a schema mapping m, and (3) a collection $(D_1, d_1), \ldots, (D_n, d_n)$ of *local* databases.

In the schema mapping, the views in the first position represent all data contributed from the local databases. The views in the second position represent materialization of the corresponding first-position views from the local databases. A user query expressed in terms of the *global schema* is translated into a query over the *global views* (the views in the first position of the schema mapping). This process is called a *query translation method*. The challenge in this translation is that data requested by the user query (and described in the global schema) may not be in *any* of the available information sources, or may appear in *several* of them (thus resulting in possible data inconsistency).

The query translation method needs to be extended to handle data inconsistency.

3 The Extended Model

3.1 Virtual Database and Schema Mapping

A *virtual database* consists of (1) a *global schema* $D=\{R_1, \ldots, R_N\}$, (2) a *schema mapping* m, and (3) a collection $(D_1, d_1), \ldots, (D_n, d_n)$ of *local* databases. The schema mapping is defined as a set of triplets (V, URL, P), where V is a projection-selection-join (PSJ)[1] view definition over the global schema D, URL is an expression that is used to materialize this view from one of (possibly remote) information sources (D_i, d_i), P is a set of the source *properties* and V, URL are equivalent.[2]

3.2 Properties

A source *property* consists of a *property name*, such as **timestamp**, **cost** or **priority**, and a numeric *property value*. Each property possesses the following characteristics.

– A native property value range. E.g. value range of **priority** might be the interval [0, 1], and values of **cost** could range between 0 and M, where M is an arbitrarily high number.
– A total order $\preceq$ on the set of the property values. Throughout the paper the term "better" will be used to denote $\preceq$ (i.e. b is better then a iff $a \preceq b$). As an example, consider the property **priority** with higher values denoting more credible values. Then $a \preceq b$ is equivalent to $a \leq b$ (i.e. an priority value 1 is better then an priority value 0.5). As another example, consider the property **cost** with higher values denoting costlier data. Then $a \preceq b$ is equivalent to $a \geq b$ (i.e. a cost value 5 is better then a cost value 15).

To facilitate comparisons between different property values, all properties are normalized as follows. Each property value is *linearly* mapped to a number in the interval [0, 1], where the worst property value is mapped to 0, and the

[1] We assume that the information in the local DBs is restricted to PSJ views
[2] Note that this definition of schema mapping extends definition in section 2.

best — to 1. Such mapping depends on the information sources participating in the integration and therefore should be determined at run-time. An example of normalization of several properties follows.

1. **Cost**. Let c_i be the cost of one of N information sources. Since 1 represents the *best* cost, and 0 — the *worst*, the normalized cost is computed as follows:
$$c = 1 - \frac{\textbf{cost}}{\sum_{i=1}^{N} c_i}$$
2. **Timestamp**. Timestamp reflects a point in time when the data were created. Let t_{now} be the timestamp of the moment when the normalization is performed. Let t_0 be some fixed timestamp in the past. Then the normalized timestamp
$$t = \frac{\textbf{timestamp} - t_0}{t_{now} - t_0}$$
3. **Priority**. If p_{max} is the highest priority among the sources, then the normalized priority for a particular source
$$p = \frac{\textbf{priority}}{p_{max}}$$

Every information source has a set of properties associated with it, but in practice, different sources may have different property sets. Therefore, a *global property set* $\mathcal{P}$ is defined, that contains all the properties from the source property sets.[3] Each source property set is augmented to $\mathcal{P}$ by adding *null* values in place of the values of the properties that are not associated with the information source.

Throughout this paper, $\mathcal{P}$ will be used instead of the original property sets.

Property Propagation The aforementioned definition of properties associated each property value in a source's global property set with the *entire* information source. All properties are assumed to be inherited by all individual tuples and all their attribute values (i.e. source properties are propagated to the level of individual attribute values within the source).

Clearly, this assumption implies that the data in every source are homogeneous with respect to every property. Note that in situations where an information source is heterogeneous with respect to a given property, it might be possible to partition the source into several disparate areas that would be homogeneous with respect to that property. These areas would consequently be treated as separate information sources.

Extended Schemas and Instances To incorporate the notion of properties into the model, the definitions of relation schemas and instances must be extended.

- A *relation schema* R=(A_1, …, A_m, P_1, …, P_k) is a sequence of *some* attribute names from $\mathcal{A}$ followed by a sequence of *all* the property names from $\mathcal{P}$.
- A *tuple* t of a relation schema R=(A_1, …, A_m, P_1, …, P_k) is an element of $dom(A_1) \cup \{null\} \times \cdots \times dom(A_m) \cup \{null\} \times dom(P_1) \cup \{null\} \times \cdots \times dom(P_k) \cup \{null\}$. t will often be written as (a, p), where a is a sequence of attribute values and p is a sequence of property values.
- A *relation instance* r of a relation schema R is a finite set of tuples of R.

[3] I.e. property names of $\mathcal{P}$ are the union of property names from all the information sources.

3.3 Extended Relational Algebra

To accommodate the aforementioned relational model extensions, the following extended relational algebra operations are introduced.

- **Extended selection**: $\overline{\sigma}_\phi(r) = \sigma_\phi(r)$
  Here ϕ is the selection predicate over the *attributes* of R (and not its properties).
- **Extended projection**: $\overline{\pi}_{A_{j_1},\ldots,A_{j_n}}(r) = \pi_{A_{j_1},\ldots,A_{j_n},P_1,\ldots,P_k}(r)$
- **Extended Cartesian product**: $r_1 \overline{\times}\, r_2 =$
  $\{(a_1, a_2, min(p_1^1, p_1^2), \ldots, min(p_k^1, p_k^2)) | (a_1, p_1^1, \ldots, p_k^1) \in r_1 \wedge (a_2, p_1^2, \ldots, p_k^2) \in r_2\}$
  Intuitively, tuples of $r_1 \overline{\times}\, r_2$ are tuples of the regular Cartesian product extended by their *combined* properties. Corresponding property values are combined by taking the minimum of the two operands.
  The reasoning behind the choice of the minimum as a combining function is as follows. Recall that, because of the existence of the total order for the property values, better property values are mapped to higher numbers from the interval [0, 1]. Therefore, selecting minimum as a combining function ensures that all the attributes in the combined tuple have *at least* the combined value for each property. Thus, property propagation is preserved throughout the newly formed tuples. Note that *null* property values are ignored by the minimum function.
- **Extended union**: $r_1 \overline{\cup}\, r_2 = r_1 \cup r_2$
- **Extended difference**: $r_1 \overline{-}\, r_2 = r_1 - r_2$
- **Property selection**: $\omega_\psi(r) = \sigma_\psi(r)$
  Here ψ is the selection predicate over the *properties* of R (and not its attributes).
- **Conflict resolution**: $resolve_{w_1,\ldots,w_k}(r) = \{(a, p) | (a, p) \in r^{res}\}$
  This operation assumes the existence of an algorithm for resolving data value conflicts.[4] Intuitively, it transforms a given relation instance r into a relation instance r^{res} in which all data value conflicts have been resolved. w_1, $\ldots$, w_k are the corresponding weights of the properties p_1, $\ldots$, p_k.

For simplicity, we will use the standard symbols for the extended relational algebra operations throughout the paper (i.e. σ will be used instead of $\overline{\sigma}$ etc.).

3.4 Contributions and Query Fragments

Recall that the extended schema mapping is a set of triplets (V, URL, P). Each triplet (V, URL, P) is called a *contribution* to the virtual database. Obviously, not all contributions are needed for every user query. To determine the contributions relevant to a given query, the following two-step process is used.

First, the sets of attributes of the query and a contribution are intersected. If the intersection is empty, the contribution is deemed not relevant. Next, the selection predicates of the query and the contribution are conjoined. If the resulting predicate is not *false* (i.e. it is satisfiable), then the contribution is considered relevant to the query.

From each relevant contribution we need to derive a unit of information suitable for populating the query answer. Such unit is termed *query fragment.*

Intuitively, to obtain a query fragment from a contribution C we need to remove all tuples and all attributes of the contribution that are not included in the query, and then add *null* values for the query attributes not present in the contribution.

[4] The algorithm will be described in subsection 5.3.

Let V be a PSJ view.[5] Assume that V's selection predicate contains an equality $A_i = A_j$ and that A_j is projected out by V's projection. For all tuples in v the values of A_j can be reconstructed from the values of A_i by adding A_j to V and the column of A_j values to v. This process, which is repeated for all equalities in V's selection predicate, is called an *enhancement* of V and its result is denoted $\overline{V}$.

Formally, a query fragment derived from a contribution $C = (V, URL, P)$ and a query $Q = \pi_{A_{j_1},\ldots,A_{j_s}} \omega_\psi \sigma_\phi (R_{i_1} \times \ldots \times R_{i_k})$ is defined as follows:

1. Let $\overline{V}$ be the enhancement of the contribution view V and let $\overline{v}$ be the enhancement of v.
2. Denote $Y = \{A_{j_1}, \ldots, A_{j_s}\} \setminus \overline{V}$ (set difference between the two schemas). Let y be an instance of Y, which has a single tuple t_{null} composed entirely of *null* values.
3. Let $\mathcal{P}$ be the global property set and let p be the tuple of property values of C.
4. A *query fragment* derived from C (denoted Q^C) is a view definition whose schema is $\{A_{j_1}, \ldots, A_{j_s}\} \cup \mathcal{P}$, and for every instance v of V, the instance of Q^C, denoted q^c, is $\omega_\psi \sigma_\phi (\pi_{A_{j_1},\ldots,A_{j_s}}(\overline{v}) \times y \times p)$.
5. The selection criteria of V is said to be *associated* with the query fragment Q^C.[6]

Example. Assume a global relation $R = (A, B, C, D, E, \mathit{timestamp}, \mathit{cost}, \mathit{priority})$. Assume a contribution $C = (V, URL, P)$ with $V = \pi_{A,B,C} \sigma_{B>0 \wedge C=E} R$, $P = (\mathit{timestamp}{:}0.7, \mathit{cost}{:}0.8, \mathit{priority}{:}1)$ and $URL =$ *"http://www.aname.com/smth.cgi?action=retrieve&ID=517"*. Consider a user query $Q = \omega_{\mathit{timestamp}>0.5} \pi_{A,B,D,E} \sigma_{C<10} R$.

Let $v =$

A	B	C
1	2	7
2	2	11
3	4	4

Since V includes an equality $C = E$, $\overline{v} =$

A	B	C	E
1	2	7	7
2	2	11	11
3	4	4	4

The query Q fragment formed from the contribution C is

$q^c =$

A	B	D	E	*timestamp*	*cost*	*priority*
1	2	*null*	7	0.7	0.8	1
3	4	*null*	4	0.7	0.8	1

Note that the column D contained in Q's projection set is not available from the contribution, and is therefore presented as a column of *null* values.

A single query fragment is constructed from each relevant contribution. The extended union of all query fragments is termed *multi-instance* of the query. The multi-instance encompasses all the information culled from the data sources in response to the user query.

From a user perspective, a query against the global database is supposed to return single consistent answer. Therefore, all possible data value inconsistencies within the multi-instance must be resolved by applying the *resolve* operation described in the following sections.

[5] In this paper we restrict views to those that do not contain comparisons across the view relations.

[6] Note that the selection criteria associated with Q^C can contain attributes not from Q.

Recall that query translation is a process in which a query over the global relations is translated into a query over the global views. Since each query fragment is a view over a global view, the multi-instance is a view over the global views as well. Hence, the construction of the multi-instance is a query translation process.

4 Data Inconsistency Detection

Often, the term *data inconsistency* is used to describe schematic differences among heterogeneous databases. A number of approaches to resolution of schema inconsistencies exist [11], [2], [7], [3], [14] that enumerate all possible translations of a user query into a query in terms of the source descriptions. To perform query translation, one needs to *map* attributes in the schemas of *local* (being integrated) databases into the global schema attributes.

The term “data inconsistency” is also used to describe inconsistencies at the level of *data representation*. Even when global and local attributes are mapped successfully, they still can be conflicting in their representations of data ([17], [11]). Although that type of conflict is seemingly based on the content and not the schema, it can be resolved by a simple conversion and therefore does not constitute a factual discrepancy between the sources.

What common in the approaches that handle the above two types of inconsistency is an implicit assumption that the *contents* of all the information sources are mutually consistent [13].

A data inconsistency exists when two objects (or tuples in the relational model) coming from different information sources are identified as *versions* of each other (i.e. they represent the same real-world object) and some of the values of their corresponding attributes differ. Note that such identification is only possible when both schema inconsistency and data representation differences have been resolved. The above definition leads us to two logical steps required for data integration process: inconsistency *detection* and inconsistency *resolution*.

4.1 Tuple Identification: Keys vs. Clustering

In the first phase of the data inconsistency detection, multi-instance tuples that are versions of each other need to be identified. The obvious approach for tuple identification would be to use *keys* of the global relations. Then, to find all tuples in the multi-instance that are versions of each other, one could construct the key of the query answer and group the multi-instance based on it.

Unfortunately, such an approach is not always possible: key attributes may not be available from the relevant contributions, or their values may be not reliable enough for identifying tuples. In that case an alternative approach based on a *tuple similarity measure* [9], [16], [12] is employed. In this approach, every tuple is mapped to a binary vector in a semantic vector space and the *similarity* between any two tuples of the multi-instance can be computed as one of the *vector coefficients*, e.g. matching coefficient, Dice coefficient, overlap coefficient, or Jaccard coefficient. The more similar two tuples are, the higher probability of them being versions of each other. Therefore, based on the tuple similarity measure, a threshold and a clustering technique, the multi-instance of the query answer is *clustered* into non-overlapping sets of tuples that are versions of each other.

4.2 Determining the Possible Areas of Inconsistency

Numerous clustering techniques exist (for example, [8], [15], [10]) that are suitable for tuple identification.[7] However, these techniques can be improved upon considerably given additional *constraining information* in the form of the selection predicates associated with query fragments.

If selection predicates of two query fragments are contradictory (i.e. their conjunct is *false*), then one of these query fragments must not contain tuples that are versions of tuples in the other fragment. Therefore, the multi-instance can be *partitioned* by the query fragment selection predicates before it is clustered, resulting in clustering being applied within the partition elements only. The partition elements are termed *slices*. By constraining the clusters to this partition, a higher level of clustering accuracy is guaranteed (note that any clustering is subject to some inaccuracy).

To illustrate partitioning, consider the following example. Let q^{C_1} and q^{C_2} be two fragments of the query Q, and let ϕ_1 and ϕ_2 denote their selection predicates, correspondingly. These two fragments divide the multi-instance of Q into three mutually exclusive slices, which are defined by the following predicates:

$$\psi_1 = \phi_1 \wedge \phi_2;\ \ \psi_2 = \phi_1 \wedge \neg\phi_2;\ \ \psi_3 = \neg\phi_1 \wedge \phi_2;$$

Obviously, only q^{C_1} could contribute tuples to the slice defined by ψ_2, and only q^{C_2} could contribute tuples to the slice defined by ψ_3. However, if the selection predicates are not contradictory, i.e., $\phi_1 \wedge \phi_2 \neq false$, then both fragments could contribute tuples to the slice defined by ψ_1. Since each fragment is free of data inconsistencies, the possibility of two versions of the same tuple appearing together exists only in this slice.

In general case, for N information sources there would be $2^N - 1$ slices defined by ψ_j. However, some of the associated ψ_j predicates might evaluate to *false*, in which case the corresponding slices would be empty.

After partitioning of the multi-instance into slices, the clustering is done within each slice. The resulting clusters are termed *multi-tuples*. Each of them contains tuples that (with certain accuracy) are versions of each other and may be visualized as a table:

Name	*Age*	*Salary*	*timestamp*	*cost*	*priority*
Smithson	38	75000	0.8	0.5	1
Smith	35	*null*	0.7	0.2	*null*
Schmidt	35	77000	0.7	0.8	1

5 Data Inconsistency Resolution

Once data inconsistencies have been detected, the next step is to attempt and resolve them, i.e. every multi-tuple should be reduced to a single tuple. This process is performed in two passes. In the first pass (subsection 5.1), the tuple versions within each multi-tuple are ranked and purged according to user-specified preferences. In the second pass (subsection 5.2), remaining values of each attribute in the multi-tuple are purged and then fused into a single value. Similarly, each multi-tuple property values are fused into a single value (one per

[7] Discussion on comparison of the clustering techniques is beyond the scope of this paper.

property). These passes are formalized and summarized in the subsection 5.3. The described procedure is the actual implementation of the aforementioned extended relational algebra operation $resolve_{w_1,\dots,w_s}$.

5.1 Objective function

Recall that data inconsistency resolution is described by an extended relational algebra operation $resolve_{w_1,\dots,w_s}$. By specifying property weights $w_1, \dots, w_s$, users indicate the importance level of the corresponding properties for the resolution process. To use this information, an *objective function* $f^{obj} = \sum_{i=1}^{s} w_i p_i$ is computed for all tuple versions in every multi-tuple. Here p_i are the property values for the tuple version, and w_i are their corresponding weights defined by the user. Then, within each multi-tuple, tuple versions are *ranked* according to the value of the objective function. Using a pre-defined threshold (or *acceptability margin*), all tuple versions with an objective function value lower than the threshold are discarded.

It is important to note, that this process does not guarantee complete resolution of every multi-tuple, since several tuple versions with an acceptably high objective function value can exist. In addition, the presence of *null* values can complicate the computation of the objective function or make it impossible to compute altogether.[8]

To fine-tune the resolution process, *resolution strategies* defined by domain experts are needed.

5.2 Resolution Strategies

Ideally, a conflict resolution strategy should be provided whenever data inconsistency is possible. However, as was shown previously, having n different data sources could result in as many as $2^n - 1$ possible areas of conflict. Providing a separate resolution strategy for every area would be quite inefficient. Instead, a single strategy is defined per each global attribute. Therefore, within each multi-tuple data inconsistency is resolved for each attribute separately (i.e. each column of values in the multi-tuple is fused into a single value). To perform such resolution, the multi-tuple is first divided into several *multi-attributes*, where each multi-attribute is a projection of the multi-tuple on one of its global attributes (see Figure 1). Then, each multi-attribute is resolved to a single attribute value and its corresponding unique property values. Finally, the resolved multi-attributes are joined together, resulting in a resolved multi-tuple.

$\bar{t} =$

Name	*Salary*	*timestamp*	*priority*
Smithson	75000	0.8	1
Smith	*null*	0.7	*null*
Schmidt	77000	0.7	1

$\downarrow$ $\downarrow$

$\overline{a_1} =$

Name	*timestamp*	*priority*
Smithson	0.8	1
Smith	0.7	*null*
Schmidt	0.7	1

$\overline{a_2} =$

Salary	*timestamp*	*priority*
75000	0.8	1
null	0.7	*null*
77000	0.7	1

Fig. 1. A multi-tuple is divided into two multi-attributes.

As noted previously, the resolution is performed for each multi-attribute separately. This process consists of two phases. In the first phase, some of the rows

[8] See subsection 5.3 for discussion on *null* values.

in the multi-attribute are *eliminated*, based either on the attribute values or the values of the properties. In the second phase, the remaining rows are *fused*, producing single values for both the attribute and the properties. Correspondingly, a resolution strategy is defined as a sequence of *elimination functions*, followed by a *fusion function*.

Each elimination function can be applied to either the attribute values or values of one of the properties within the multi-attribute. Examples of such elimination functions are min and max. Consider the multi-attribute $\overline{a_2}$ from the previous example. The elimination functions can be $max(timestamp)$ or $min(timestamp)$, $max(priority)$ or $min(priority)$, and $max(Salary)$ or min $(Salary)$. Each function is applied in its turn (according to its place in the sequence) to the correspondent column of $\overline{a_2}$, eliminating all but one value. However, this application can still result in multiple rows (e.g. several values of $Salary$ have the maximal priority 1). Therefore, at the end of the elimination phase the multi-tuple may still be unresolved. Note also, that functions other than min or max can be used. For example, $over\_average$, $top\_five\_percent$ and $within\_standard\_deviation\_of\_the\_mean$.

The fusion function is applied both to the values of the global attribute and the values of the properties, resulting in a single resolved value for each one of them. Examples of such fusion functions are $random$ and avg. Consider the multi-attribute $\overline{a_2}$ from the previous example. Application of $random$ immediately results in a resolved multi-attribute, for example (77000, 0.7, 1). The avg is applied to the values of attribute $Salary$, resulting in 76000 as its resolved value. The property values, however, cannot be simply averaged, as it would not reflect the correct property values associated with the fused attribute value 76000. If a conservative approach is adopted, the resolved values for each property are calculated as minimum of all the corresponding property values, yielding (76000, 0.7, 1) as the resolved multi-attribute. Note, that functions other then $random$ or avg can be used. For example, $mean$, $avg\_without\_extreme\_values$, or, in general, a linear combination of the conflicting values.

To give users full control over the process of inconsistency resolution, a flexible yet powerful resolution statement is needed, which incorporates the experts' *domain knowledge*. So the inconsistency resolution statement for a given global attribute is defined as follows.[9]

for A_i
[**choose** $f_1(E_1)$**, ...,** $f_n(E_n)$] — optional clause
fuse by g

A_i is a global attribute name. Each of E_i is either empty (and therefore indicates an operation on the global attribute values) or one of the property names in the global property set. f_1, ..., f_n are elimination functions,[10] and g is a fusion function.

A resolution statement is necessary for every global attribute.

When every multi-attribute of a multi-tuple is resolved, the resolved multi-tuple needs to be reconstructed, so the resolved multi-attributes are joined back together by taking their Cartesian product. According to the definition of the extended Cartesian product, property values of different multi-attributes within the resolved multi-tuple are fused by adopting their minimum as the fusion result.

[9] The statement is SQL-like but lies outside of the SQL.
[10] Applied in the order they are written in.

The resolution process is done for every multi-tuple, resulting in a single *resolved* tuple. The set of the resolved tuples is then presented to the user as a single-valued consistent query answer.

Example. Consider the following query: $Q = \delta_{priority>0.8}\ \sigma_{Position=''Manager''}\ \pi_{Name,Age,Salary}\ Employees$. Let $\bar{t}$ be a multi-tuple to be resolved.

$\bar{t} =$

Name	*Age*	*Salary*	*timestamp*	*cost*	*priority*
Smithson	35	77000	0.8	0.5	1
Smith	38	*null*	0.7	0.2	*null*
Schmidt	35	75000	0.7	0.8	1

Assume that the following resolution statements are in effect.

for $Name$ **choose** *min*(*timestamp*) **fuse by** *random*
for *Age* **choose** *min*() **fuse by** *average*
for *Salary* **choose** *max*(*cost*) **fuse by** *random*

The *resolved* tuple:

$t =$

Name	*Age*	*Salary*	*timestamp*	*cost*	*priority*
Schmidt	35	75000	0.7	0.2	1

Here, minimum timestamp yields property vector {0.7, 0.5, 1}, minimum age — {0.7, 0.2, 1}, and maximum salary — {0.8, 0.2, 1}. The resulting property vector is {0.7, 0.2, 1}.

5.3 The Overall Resolution Procedure

The methodology described above is formalized in the following procedure for resolving data inconsistencies. It implements the extended relational operation $resolve_{w_1,\ldots,w_s}$.

1. Each property name p_i is assigned a *weight* or importance factor w_i.
2. An *acceptability margin* (a number between 0 and 1) b is defined.
3. For each of the tuple versions t_j an *objective function* f^{obj} is constructed:[11]
$$f_j^{obj} = \sum_{i=1}^{s} w_i p_i$$
4. Tuple versions in a multi-tuple are *ranked* based on the value of the objective function.
5. Tuple versions with an objective function value less then $(1-b)\max_j(f_j^{obj})$ are discarded in each multi-tuple.
6. According to the inconsistency resolution statement in effect, attribute values are purged, then the surviving *non-null*[12] values of every attribute in the multi-tuple are *fused* together, resulting in a single value.
7. Consistent with the same statement, different values of every property in the multi-tuple are *fused* together, resulting in a single property value.[13] If a conservative approach is chosen, the value is calculated as the minimum of the participating property values.

[11] A situation when some of the property values are *nulls* will be examined further in the text.
[12] If all attribute values are *nulls* then the fusion value is set to *null.*
[13] If all attribute values are *nulls* then each fusion property value is set to the minimum of all the corresponding property values

Null Values Computation of the objective function value on a particular tuple version is complicated by the possibility of *null* property values (which means that the corresponding properties are not available from the contributing information source). When such situations arise, the objective function is computed on a *maximal subset* of the property values that are not *null* for every tuple version in the multi-tuple.

For example, let $\bar{t}$ be a multi-tuple to be resolved, and let objective function $f^{obj} = 0.3 timestamp + 0.5 cost + 0.2 priority$.

$\bar{t} =$

Name	*Age*	*Salary*	*timestamp*	*cost*	*priority*
Smithson	35	77000	0.8	0.5	1
Smith	38	*null*	0.7	0.2	*null*
Schmidt	35	75000	0.7	0.8	1

The maximal *non-null* subset of the properties is $\{timestamp, cost\}$ (since the second tuple version in $\bar{t}$ contains *null* for the *priority* property). Therefore, the objective function is computed as $f^{obj} = 0.3 timestamp + 0.5 cost$.

Tuple versions for which the objective function cannot be computed are retained during the first pass. Alternatively, these tuple versions could be discarded from the multi-tuple and the objective function should be computed for the remaining tuple vesrions.

6 Implementation

The methodology presented in this paper have been implemented in a prototype system available at *http://www.ise.gmu.edu/∼mltplx*, which is called Multiplex-II. Figure 2 illustrates its overall architecture.

The Multiplex-II makes use of a server-client architecture. The server is implemented in Java and contains the core functionality described in the previous sections. Clients can connect to the server using simple line-based protocol. Each client passes to the server the name of a database which the client's user wishes to query and the query itself. Then the server processes the query and returns its result to the client, which passes it to its user in a chosen format.

The core of the Multiplex-II consists of the following functional blocks: query parser, query translator, view retriever, fragment factory, conflict detection module, conflict resolution module and query processor.

A client using Graphic User Interface is included in the implementation of the Multiplex-II. The web-enabled GUI supports both direct input of the query as a text and a wizard-like query construction through use of the Query Assistant, which allows users to create queries using an intuitive visual interface.

The Multiplex-II GUI also incorporates a database management tool, which allows for addition, deletion and modification of the system databases. Authorized users can create virtual multidatabases, add, delete, rename the multidatabase relations and attributes, and plug in the existing data sources that they need to integrate.

7 Conclusions

This paper addresses the data integration problem from a new perspective. Unlike most of the approaches in the area of information integration, the approach introduced in this paper acknowledges the existence of inconsistencies not just among different schemas or data representations, but data themselves. An extension to the relational data model is proposed, making use of meta-data such

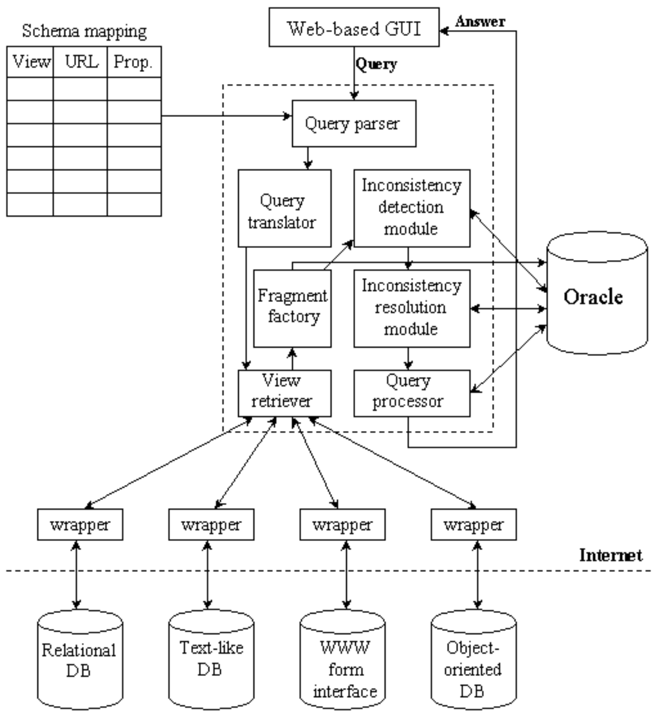

Fig. 2. Architecture of the Multiplex-II

as *properties* of the information sources in user queries, as well as in the data value conflict resolution. A full set of extended relational operations over the extended relations is also developed.

The process of data integration described in this paper consists of the following phases: construction of the data blocks such that their extended union constitutes the query result, data conflict detection and data conflict resolution.

In the conflict detection phase a clustering technique is used for identification of conflicting objects. It is shown that a technique can be improved considerably by constraining the clustering to smaller areas. These areas of possible conflict are detected by using another type of meta-information available from the sources: source descriptions in terms of the virtual database schema.

For the conflict resolution phase, a flexible algorithm is offered. The algorithm is guided by user-defined importance (or *weights*) of properties and by expert-defined resolution strategies that incorporate the domain knowledge.

8 Future Research

During the fusion phase of the information integration algorithm attributes from different *versions* of tuples coming from different information sources can end up together in the same *resolved* tuple. That can be undesirable in some situation. The following could serve as an example: there exist two different versions of address information for a particular person; the resolution mechanism should not accept a city name from one of the versions and a zip code from the other, as it would result in incorrect information. In this example, city name and zip code together constitute an *unbreakable* sub-tuple of the address. The data integration algorithm has to be extended to handle such unbreakable (sub-)tuples correctly.

Another interesting direction for the further development is a problem of automatic or semi-automatic *discovery* of new information sources that can be integrated into the system. Obviously, the Internet offers many opportunities in this area.

References

1. S.Agarwal, A.M.Keller, G.Wiederhold, and K.Saraswat "Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases", ICDE, 1995: 495–504.
2. Y.Arens, C.A.Knoblock, and C.N.Hsu "Query Processing in the SIMS Information Mediator", Advanced Planning Technology, Austin Tate, AAAI Press, Menlo Park, CA, 1996.
3. S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom "The TSIMMIS Project: Integration of Heterogeneous Information Sources", IPSJ, 1994: 7–18.
4. M.Arenas, L.E.Bertossi, J.Chomicki "Consistent Query Answers in Inconsistent Databases", PODS, 1999: 68–79.
5. E.F.Codd "Extending the Database Relational Model to Capture More Meaning", In Proceedings of ACM TODS, December 1979: Volume 4, 397–434.
6. L.G.DeMichiel "Resolving database incompatibility: an approach to performing relational operations over mismatched domains", TKDE 1(4), 1989.
7. O.M.Duschka, and M.R.Genesereth "Infomaster - An Information Integration Tool", in proceedings of the International Workshop "Intelligent Information Integration" during the 21st German Annual Conference on Artificial Intelligence, KI-97. Freiburg, Germany, September 1997.
8. M.A.Hernandez, S.J.Stolfo "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem", Data Mining and Knowledge Discovery 2(1), 1998: 9–37.
9. M.James "Classification Algorithms", John Wiley and Sons, 1985.
10. L.Kaufman and P.J.Rousseeuw "Finding Groups in Data: an Introduction to Cluster Analysis", John Wiley and Sons, 1990.
11. A.Y.Levy, A.Rajaraman, and J.J.Ordille "Querying Heterogeneous Information Sources Using Source Descriptions", In Proceedings of VLDB Conference, 1996.
12. E-P.Lim, J.Srivastava, S.Shekhar "Resolving Attribute Incompatibility in Database Integration: An Evidential Reasoning Approach", ICDE, 1994: 154–163.
13. A.Motro "Multiplex: A Formal Model for Multidatabases and Its Implementation.", NGITS, 1999: 138–158.
14. T.Milo, S.Zohar "Using Schema Matching to Simplify Heterogeneous Data Translation", VLDB, 1998: 122-133.
15. E.M.Rasmussen "Clustering Algorithms.", Information Retrieval: Data Structures and Algorithm, 1992: 419-442.
16. H.Romesburg "Cluster Analysis for Researchers", E. Rober Krieger, 1990.
17. A.Rosenthal, E.Sciore "Description, Conversion, and Planning For Semantic Interoperability", DS-6, 1995: 140-164.
18. V.S.Subrahmanian, S.Adali, A.Brink, R.Emery, J.J.Lu, A.Rajput, T.J.Rogers, R.Ross, and C.Ward "HERMES: Heterogeneous Reasoning and Mediator System", unpublished paper, 1994.
19. F.S-C.Tseng, A.L.P.Chen, and W-P.Yang "A probabilistic approach to query processing in heterogeneous database systems", RIDE-TQP, 1992: 176–183.