Use of Meta-data for Value-level Inconsistency Detection and Resolution During Data Integration

Philipp Anokhin
Dept. of Information and Software Engineering, George Mason University
Fairfax, VA 22030, USA

and

Amihai Motro
Dept. of Information and Software Engineering, George Mason University
Fairfax, VA 22030, USA

Abstract

This paper addresses the data integration problem: there exists a collection of autonomous heterogeneous information sources that need to be integrated; users want to be able to query the collection transparently and to get a single, unambiguous answer. The sources may conflict with each other on three levels: their *schemas*, *data representation*, or *data themselves*. One has to resolve the conflicts that may arise during the integration to get the single answer to a query. Most of the approaches in this area of research resolve inconsistencies among different schemas and data representations, and ignore the possibility of data value-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 paper that consists of three phases: (1) query result construction, (2) data conflict detection and (3) data conflict resolution. An improvement to data clustering techniques in the conflict detection phase is presented in the paper. It uses another type of meta-data available from the sources (source descriptions in terms of the virtual database schema) to narrow down the areas of possible data conflicts. For the last phase, a flexible data-level conflict resolution algorithm is offered, which incorporates both *content-* and *property-based* approaches. The algorithm is guided by user-defined priorities of properties and by domain-based *resolution strategies*.

Keywords: data integration, heterogeneous autonomous sources, source properties, meta-data, inconsistency.

1 Introduction and Related Work

This paper addresses the data integration problem: there exists a collection of heterogeneous information sources that need to be integrated; users want to be able to query the collection transparently and to get a single, unambiguous answer. The sources may conflict with each other on different levels:

- **Schema**, i.e. different data models or different schemas within the same data model.
- Data **representation**, e.g. data in different languages, measurement systems etc.
- Data **value**, i.e. there is a factual discrepancy between several sources in the values for the same object.

Note that at each level a conflict can be recognized only when the previous level conflict is resolved.

Numerous approaches exist that resolve the schema inconsistencies among heterogeneous databases by enumerating all possible *translations* of a user query into a query over the source descriptions. The most optimal translation (either by the retrieval cost or the number of the participating sources) is chosen and materialized as an answer to the original query. To perform such translation, one needs to *map* attributes in the schemas of *local* (being integrated) databases into the global schema attributes. This is not a trivial task, since the corresponding attributes may have different names, the possible mappings are not always one to one and can often be created only with a limited accuracy. For example, the system described in [2] resolves schema heterogeneity by use of *semantic similarity*. It has a knowledge base for each particular domain it is modeling, which is essentially a graph with terms as nodes and fuzzy binary relationships as edges. Each term is matched to a *spanning tree* in this graph (strongest possible in terms of degree of "fuzziness"). Then the resulting trees are unified (merged together) allowing for translating of a set of terms into another set of terms in the knowledge base.

A good number of approaches also deal with different data representations in information sources. For example, [8] defines a concept of *Semantic Interoperability Service* (SIS) in a distributed object management environment. SIS consists of several functional components: argument describers, conversion

functions and a planner. Data representation inconsistences are resolved by attaching additional properties to the integrating sources: *currency* to perform currency conversion, *distance_unit* for conversion of measurement units etc.

Several approaches try to resolve the data-level conflicts using probabilistic information. For example, [9] uses *partial values* to represent uncertainty of the information. A partial value is a set of values, where each value in the set is assigned a probability of being the correct value. Selection and join operations are extended for use with this model. They discard tuples which, with certain probability threshold, do not satisfy the query conditions. [5] employs Dempster-Shafer theory of evidence to incorporate uncertainty in the source databases. Unlike [9], this approach allows assignment of properties not only to individual values in the attribute value set, but to subsets of values as well. An extention to the relational model and relational operations is proposed. The approach uses an extended *closed world assumption*.

However, some important data-level conflict resolution problems have not been addressed by the aforementioned works:

- Probability-based inconsistency resolution is only partial: instead of a single answer, a set of values with attached probabilities is given.
- Many of the approaches assume that there is no inconsistency present, just uncertainty.
- Probabilistic information has to be attached to every datum in an information source and therefore is relatively rare, especially for web-based sources.
- Users cannot influence the resolution process to get the answer with the highest quality (where "quality" is defined according to the user).

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 a potential user with some meta-data in form of the detailed source descriptions, which allows users to make judgement about relevance of a particular information source to the user's goals. The examples of such meta-data called *source properties* include the following:

- **Timestamp**: time of the source creation.
- **Cost**: either retrieval cost or amount of money paid for the information access.
- **Clearance**: clearance level needed to access the source.
- **Accuracy**: if some probabilistic information is present.
- **Authority**: source preferences supplied by the user or a group of experts.
- Etc.

Note: We assume that every datum in the source inherits the properties of the source.

In this paper we extend the classic relational model and the notion of a database query to include properties of the information sources. This extension gives ground to a flexible data integration technique that consists of three phases: query result construction from *query fragments*, data conflict detection and data conflict resolution. An improvement to known data clustering techniques in the data conflict detection phase is presented, which uses another type of meta-data available from the sources to narrow down the areas of possible data conflicts. For the data conflict resolution phase, a flexible algorithm is offered, which incorporates both *content-* and *property-based* approaches. The algorithm is guided by user-defined priorities of properties and by domain-based *resolution strategies*.

2 Extended Relational Model[1]

2.1 Properties and Databases

A *property* consists of a *property name*, such as timestamp, cost, priority etc., and a numeric *property value*. We assume that all properties are normalized: each property value is a number in the interval [0, 1], where the "worst" property value is mapped to 0, and the "best" — to 1.[2]

Let $R_1, \ldots, R_N$ be relation schemes, and $r_1, \ldots, r_N$ – the corresponding relation instances. Each relation schema R_i is defined as a set $\{A_1, \ldots, A_m, P_1, \ldots, P_n\}$, where $\{A_1, \ldots, A_m\}$ is a set of the relation attribute names, and $\{P_1, \ldots, P_n\}$ is a set of property names. Each relation instance is defined as a set of *tuples* $t = \{a_1, \ldots, a_m, p_1, \ldots, p_n\}$, where $a_i \in dom(A_i)$ and $p_i \in dom(P_i)$. Then (D, d) is a *relational database*, where $D = \{R_1, \ldots, R_N\}$, and $d = \{r_1, \ldots, r_N\}$.

2.2 Extended Relational Algebra Operations

The following extended relational algebra operations are introduced:

- **Extended selection** $\overline{\sigma}_\phi(R) := \{(a,p) | (a,p) \in R \wedge \phi(a) = true\}$
- **Extended projection** $\overline{\pi}_{A_{j_1},\ldots,A_{j_k}}(R) := \pi_{A_{j_1},\ldots,A_{j_k},P_1,\ldots,P_n}(R)$
- **Property selection** $\omega_\psi(R) := \{(a,p) | (a,p) \in R \wedge \psi(p) = true\}$. Selection condition ψ is a conjunct of atomic boolean expressions.
- **Extended Cartesian product** $R_1 \overline{\times} R_2 := \{(a_1, a_2, p_1, \ldots, p_s) | (a_i, p_1^i, \ldots, p_s^i) \in R_i \wedge p_j = \min_{i=1}^{2}(p_j^i)\}$. The minimum of two properties with the same name is taken as the resulting property value. Note that *null* property values are ignored by *min*.
- **Extended union** $R_1 \overline{\cup} R_2 := R_1 \cup R_2$, where R_1 and R_2 are union-compatible relations.

[1] The model described in this section is an extention of a multidatabase model in [6].

[2] I.e. zero cost is assumed to be equal to 1, as well as a most recent time-stamp.

- **Extended intersection** $R_1 \overline{\cap} R_2 := R_1 \cap R_2$, where R_1 and R_2 are union-compatible relations.
- **Conflict resolution** $resolve_{w_1,\ldots,w_s}(R) := \{(a, p_1, \ldots, p_s) | (a, p_1, \ldots, p_s) \in R^{res}\}$. R^{res} is R with all data value-level conflicts resolved according to weights w_i of p_i. The conflict resolution algorithm will be explained in detail in further sections.

The extended relational algebra operations can be combined to form the extended relational algebra expressions. For simplicity, the line over the relational operation symbols throughout the paper will be omitted (i.e. σ will be useed instead of $\overline{\sigma}$ etc.).

2.3 Views and Queries

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

Note that the above definitions allow for nulls in relation instances. This requires appropriate extensions to the model to determine the results of comparisons that involve nulls. Codd's three-valued logic ([1]) can be adopted for that purpose, in which comparisons that involve nulls evaluate to the value *maybe*.

2.4 View Equivalence and Schema Mappings

Consider a database (D, d). Let D' be a database schema whose relation schemas are defined as views of the relation schemas of D. The database schema D' is said to be *derived* from the database schema D. Let d' be the database instance of D' which is the extension of the views D' in the database instance d. The database instance d' is said to be *derived* from the database instance d. Altogether, the database (D', d') is a *derivative* of the database (D, d).

Let (D_1, d_1) and (D_2, d_2) be two derivatives of a database (D, d). A view V_1 of D_1 and a view V_2 of D_2 are *equivalent*, if for every instance d of D the extension of V_1 in d_1 and the extension of V_2 in d_2 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. A *schema mapping* (D_1, D_2) is a collection of m pairs (V_{i1}, V_{i2}), where for any $i = 1, \ldots, m$ V_{1i} is a view of D_1, V_{i2} is a view of D_2 and V_{i1} is equivalent to V_{i2}.

2.5 Virtual Multidatabase

Assume that there exists a hypothetical database (D^*, d^*) that represents the real world, and the schema and instance of this ideal database are perfectly correct. Assume a *global* schema D and a collection $(D_1, d_1), \ldots, (D_n, d_n)$ of *local* databases such that the schemas D and $D_1, \ldots, D_n$ are derivatives of the real-world schema D^*, but the instances $d_1, \ldots, d_n$ are not necessarily derivatives of the real-world instance d^*. Therefore, we assume that all differences among database *schemas* $D_1, \ldots, D_n$ are *reconcilable*, but we allow for the possibility of *irreconcilable* differences among the database *instances* $d_1, \ldots, d_n$.

A *virtual multidatabase* is:

1. A global schema D.[3]
2. A collection $(D_1, d_1), \ldots, (D_n, d_n)$ of local databases.
3. A global schema mapping $M = \cup_{i=1}^{n}(D, D_i)$.

The above definition of a multidatabase provides a framework for integration of heterogeneous information sources. The first item defines the integrating schema of a multidatabase, the second item defines the local information sources being integrated, and the third item defines a mapping from the global schema to the schemas of the sources. For illustration purposes, we will view pairs in M as triplets (V, URL, P), where V is a view on the global schema, URL describes an equivalent to V view on a local source and provides means for retrieval of the view instance, and P is a set of properties associated with URL.[4] We also assume that there exists a *global property set* $\overline{P}$, such that the set of all its property names is a union of all the property names in M. Each property set P in M is enhanced to $\overline{P}$ by adding *nulls* in place of the values of the properties that are not in P. From now on, we will use this extension instead of the original property sets.

2.6 Contributions and Query Fragments

Each triplet (V, URL, P) in the schema mapping represents a *contribution* of the corresponding information source to the system. However, depending on the user query, only parts of the possible contributions may be needed to create the query answer, or some parts of the answer may not be found in the available contributions. Therefore, for each of the contributions, we need to derive from it a unit of information suitable for populating the query answer. The units are termed *query fragments*.

Formally, we define a query fragment derived from a contribution $C = (V, URL, P)$ as follows:

1. Let $Q = \pi_{A_{j_1},\ldots,A_{j_s}} \delta_\psi \sigma_\phi (R_{i_1} \times \ldots \times R_{i_k})$ be the user query.
2. Denote $Y = \{A_{j_1}, \ldots, A_{j_s}\} \setminus V$ (set difference between the two schemas).
3. Let y be an instance of Y, which has a single tuple t_{null} composed entirely of *null* values.
4. A *query fragment* derived from C (denoted Q_C) is a view definition whose schema is

[3] There is no global instance, therefore the multidatabase is *virtual*.

[4] Recall our assumption that the property set of the entire source is the same as for every datum in the source.

$\{A_{j_1}, \ldots, A_{j_s}\} \times \overline{P}$, and for every instance v of V, instance of Q_C denoted q_C is $\delta_\psi \sigma_\phi(\pi_{A_{j_1},\ldots,A_{j_s}}(v) \times y \times p)$. Here v is a materialization of V from URL, and p is a set of values of properties from P.

Simply put, to obtain a query fragment from a contribution C we remove all the attributes of C that are not included in the query and add *null* values for the query attributes not present in C.[5] Figure 1 illustrates construction of query fragments from two contributions: $C_1 = (V_1, URL_1, P_1)$ and $C_2 = (V_2, URL_2, P_2)$.

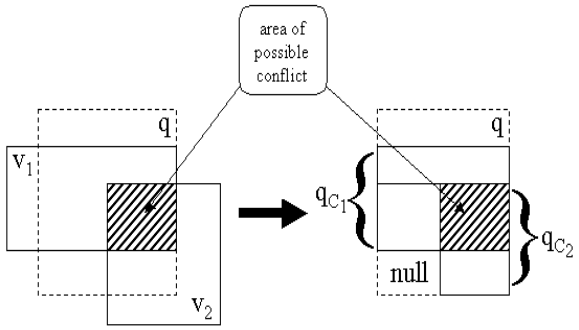

Figure 1: Constructing query Q fragments from contributions C_1 and C_2

Example. Assume the following.

- $R = (A, B, C, D, timestamp, cost, priority)$.
- $Q = \delta_{timestamp>0.5}\sigma_{C<10}\pi_{A,B,D}R$.
- $V = \sigma_{B>0}\pi_{A,B,C}R$.
- $P = (timestamp, cost, priority)$.
- $URL =$"http://site.com/retrieve.cgi?ID=517".
- $C = (V, URL, P)$. $p = (0.7, 0.8, 1)$.

$v =$

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

Then

$q_C =$

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

We term the union of all non-empty query fragments as *multi-instance* of the query. Note that from a user perspective, a query against the global database is supposed to return single consistent answer. Therefore, all possible data value-level inconsistencies within the multi-instance must be resolved by applying the *resolve* operation described in the following sections.

[5]Note that the selection criteria of q_C can contain attributes not from Q

According to the above definitions, each query fragment Q_C is a function over a view V in terms of the global relations, where $C = (V, URL, P)$ is a contribution from which Q_C was derived. In turn, the initial query Q in terms of the global relations is translated into a function over the available query fragments $Q_{C_1}, \ldots, Q_{C_k}$: $Q = resolve_{w_1,\ldots,w_s}(\cup_{i=1}^{k} Q_{C_i})$. Therefore, Q is eventually translated into a query over the views in terms of the global relations.

3 Data Value-level Inconsistency Detection

We say that data value-level inconsistency exists when two or more objects (or tuples in relation 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.[6] The above definition leads us to two logical steps required for data integration process: data inconsistency *detection* and *resolution*.

3.1 Tuple Identification: Keys vs. Clustering

As a first phase of the data inconsistency detection, 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, we could construct the key of the user query and group the multi-instance based on it.

Unfortunately, such an approach is not always possible: key attributes may not be available from the existing contributions, or their values can be not reliable enough for identifying tuples. In that case we are forced to use an alternative approach based on *tuple similarity measure* [4, 7, 5]. The more *similar* two tuples are, the higher probability of them being versions of each other. Then, based on the tuple similarity measure and a certain threshold, we could *cluster* the versions from all the query fragments together.

3.2 Determining the Possible Areas of Inconsistency

Numerous clustering techniques exist (for example, [3]), and exact details of clustering is not within the scope of this paper. However, these techniques can be considerably improved upon in the presence of additional *constraining information*: selection conditions associated with the query fragments.

By examining the selection conditions, the multi-instance can be *partitioned* before the clustering takes place, resulting in clustering being applied within the partition subsets only. Hence, we guarantee a higher level of *clustering accuracy* (note that any clustering is subject to some inaccuracy). Additionally, clustering that conforms to a partition is more *efficient*, since clustering a set of n tuples is more computationally

[6]Note that such identification is only possible when both schema inconsistency and data representation differences are resolved.

expensive than clustering k subsets of cardinality n_i ($\Sigma_{i=1}^{k} n_i = n$).

Example. Let q_1, q_2 be two fragments of the query Q, and ϕ_1, ϕ_2 — their corresponding selection conditions. These two fragments divide Q into three mutually exclusive *slices*, which are defined by applying the following predicates to each of the fragments:

$$\begin{array}{lccc} \psi_1: & \phi_1 & \wedge & \phi_2 \\ \psi_2: & \phi_1 & \wedge & \neg\phi_2 \\ \psi_3: & \neg\phi_1 & \wedge & \phi_2 \end{array}$$

Obviously, only q_1 could contribute tuples to the slice defined by ψ_2, and only q_2 could contribute tuples to the slice defined by ψ_3. However, if the selection conditions are not contradictory, i.e., $\phi_1 \wedge \phi_2 \neq false$ (three valued logic is used), then both fragments could contribute tuples to the slice defined by ψ_1. Hence, the possibility of two versions of the same tuple appearing together exists only in this slice. In this example of two contributions, each resulting slice is associated with either one or two sets of tuples. Note that in general case for N information sources there are $2^N - 1$ slices defined by ψ_j, but some of the ψ_j predicates may evaluate to *false*, in which case the corresponding sets of tuples will be empty.

Whichever mechanism is used for tuple identification, it is done within single slice. We call the resulting clusters *multi-tuples*. Each of them contains tuples that (with certain precision) are versions of each other.

4 Inconsistency Resolution

4.1 Resolution Strategies

A decision on the resolution of inconsistencies among different versions of tuples can be made either based on the properties of the versions, or their content.

Ideally, a conflict resolution strategy should be provided whenever a possibility of data inconsistency is present. However, as we showed previously, given n different data sources we could have as many as $2^n - 1$ possible areas of conflict. Trying to provide a separate resolution strategy for every one of them would be quite inefficient. Instead, we define one strategy per each global *domain*.[7]

Each resolution strategy is defined by a sequence of aggregate functions, which begins with *multi-valued* functions that do not guarantee single value as their result — such as *minimum* or *maximum*, and ends with a *single-valued* function: *average* or *random*. The functions can operate either on the conflicting attribute values (then the strategy is called *content-based*) or on their corresponding properties (then the strategy is called *property-based*). Most general inconsistency resolution strategies should intermix functions operating on attributes and properties. A *resolution statement* defining resolution strategy is given below.

[7]We assume that all the comparisons within user queries are between attributes of the same domain

for	A_{i_1}, ..., A_{i_m}	
[**choose**	$f_1(E_1)$, ..., $f_n(E_n)$]	— optional clause
fuse by	g	

A_{i_1}, ..., A_{i_m} are global attribute names that belong to the same domain. Each of E_i is either empty (and therefore indicates a content-based operation) or one of the property names in the global property set. f_1, ..., f_n are multi-valued aggregation functions,[8] and g is a single-valued content-based aggregation function. A resolution statement is necessary for every global attribute and may be supplied by the system or defined by user at run-time.

An application of a resolution strategy is a two phase process. In the first phase, multi-valued functions are applied to the conflicting attribute values. We term it *elimination* phase. However, this application can still result in multiple values (e.g. several attribute values can have the maximal timestamp). During the second, *fusion* phase, single attribute value is constructed by "fusing" together all the attribute values that "survived" elimination phase.

An application of a conflict resolution strategy may result in different property values for each attribute value in a resolved tuple. Similar to the conflicting attribute values, their corresponding property values also have to be fused. For the property fusion we adopt the conservative approach, i.e. minimum of the participating property values is taken as a result. Note that property fusion depends on the attribute fusion: if one of the conflicting attributes is eliminated, then property values of this attribute value do not participate in the property fusion.

For each property the extended data model mandates single property value per tuple. Therefore, property values of different attributes within resolved tuple are also fused by taking minimum of the values, based on the following. Let a, b be the resolved attribute values, and $\{p(a)_1, p(a)_2, p(a)_3\}$, $\{p(b)_1, p(b)_2, p(b)_3\}$ — their corresponding properties. Then the resolved tuple $(a, b, p_1, p_2, p_3) = (a, p(a)_1, p(a)_2, p(a)_3) \times (b, p(b)_1, p(b)_2, p(b)_3) = (a, b, min(p(a)_1, p(b)_1), min(p(a)_2, p(b)_2), min(p(a)_3, p(b)_3))$ according to the definition of the extended Cartesian product.

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

Example. Let $Q = \delta_{priority>0.8}\ \sigma_{Position='Mgr'}\ \pi_{Name,Age,Salary}(Employees)$ be the user query. Let the resolution statements in effect be:

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

Let the following be the multi-tuple $\overline{t}$ to be resolved:

[8]Applied in order they are written in

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

Then the *resolved* tuple t is

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}.

4.2 Resolution Algorithm

To formalize the methodology described above we propose the following algorithm for resolving data inconsistencies:

1. Within each multi-tuple, tuple versions that do not satisfy the user query are discarded.[9]
2. Each property name is assigned a *weight* w_i.
3. An *acceptability margin* b (a number between 0 and 1) is defined.
4. For each of the remaining tuple versions t_j an *objective function* f^{obj} is constructed: $f_j^{obj} = \Sigma_{i=1}^{m} w_i * p_i$
5. Tuple versions in a multi-tuple are *ranked* based on the value of the objective function.
6. Tuple versions with an objective function value less then $(1-b) * max_j(f_j^{obj})$ are discarded in each multi-tuple.
7. According to the inconsistency resolution statement in effect, different *non-null*[10] values of every attribute in the multi-tuple are *fused* together, resulting in a single value.
8. Consistent with the same statement, different values of every property in the multi-tuple are *fused* together, resulting in a single property value.[11] If a conservative approach is chosen, the value is calculated as the minimum of the participating property values.

5 Conclusion

This paper addresses the data integration problem from a new perspective. Unlike most of the approaches in this area of research, we acknowledge 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 called properties of the information sources.

We presented a process of data integration which consists of three phases: (1) construction of the data blocks such that their extended union constitutes the query answer, (2) data conflict detection and (3) data conflict resolution. In the conflict detection phase a clustering technique is used for identification of conflicting objects. We showed that the technique can be improved considerably by constraining the clustering to smaller areas of possible conflict, which are detected by using another type of meta-data available from the sources: source descriptions in terms of the virtual database schema. For the data resolution phase, we offered a flexible conflict resolution algorithm, which uses both content and properties of the information sources for inconsistency resolution. The algorithm is guided by user-defined priorities of properties and by domain-based resolution strategies.

Possible directions for further research include dealing with sources with heterogeneous properties (different properties for the different source parts), automatic discovery of relevant sources for integration into the virtual database, and automatic generation of the conflict resolution strategies according to the particular properties being used.

[9] Again, three-valued logic is employed. A tuple is discarded if the selection criteria evaluates to $false$ on the tuple

[10] If all attribute values are *nulls* then the fusion value is set to *null*.

[11] If all attribute values are *nulls* then each fusion property value is set to the minimum of all the corresponding non-null property values

References

[1] E.F.Codd "Extending the Database Relational Model to Capture More Meaning", In Proceedings of ACM TODS, December 1979: Volume 4, 397–434.

[2] P.Fankhauser, and E.J.Neuhold "Knowledge Based Integration of Heterogeneous Databases", DS-5, 1992: 155–175.

[3] 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.

[4] M.James "Classification Algorithms", John Wiley and Sons, 1985.

[5] E-P.Lim, J.Srivastava, S.Shekhar "Resolving Attribute Incompatibility in Database Integration: An Evidential Reasoning Approach", ICDE, 1994: 154–163.

[6] A.Motro "Multiplex: A Formal Model for Multidatabases and Its Implementation.", NGITS, 1999: 138–158.

[7] H.Romesburg "Cluster Analysis for Researchers", E. Rober Krieger, 1990.

[8] A.Rosenthal, E.Sciore "Description, Conversion, and Planning For Semantic Interoperability", DS-6, 1995: 140-164.

[9] 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.