

Conflict Handling Strategies in an Integrated Information System

Jens Bleiholder and Felix Naumann
Institut für Informatik, Humboldt-Universität zu Berlin
Unter den Linden 6, D-10099 Berlin, Germany
{bleiho|naumann}@informatik.hu-berlin.de

Abstract

Integrated information systems provide users and applications with a unified view of heterogeneous data sources. To provide a single consistent result for every object represented in these data sources, data fusion is concerned with resolving data inconsistencies present in and among the sources. We present a classification of conflict resolution strategies and show how these are realized using conflict handling functions. A catalog of such functions is given, together with a description of some of their properties. We further show how the functions are used within an integrated information system, the Humboldt-Merger (HumMer).

1 Data Fusion

Integrated (relational) information systems provide users with a unified view of heterogeneous data sources. Querying the underlying data sources, combining the results, and presenting them to the user is performed by the integration system.

We assume an integration scenario with a three step data integration process as shown in Figure 1. In such a scenario, when multiple, heterogeneous sources are to be integrated into a single and consistent view, at least the following three steps need to be performed: First, one needs to identify corresponding attributes that are used to describe the information items in the source. The result of this step is a *schema mapping* that is used to transform the data present in the sources into a common representation (renaming, restructuring). Second, the different objects that are described in the data sources need to be identified and aligned. In this way, using *duplicate detection* techniques, multiple, possibly inconsistent representations of same real world objects are found. Third, as a last step, the duplicate representations need to be combined and fused together into a single representation while inconsistencies in the data need to be resolved. This last step is referred to as *data fusion* and is the main focus in this paper.

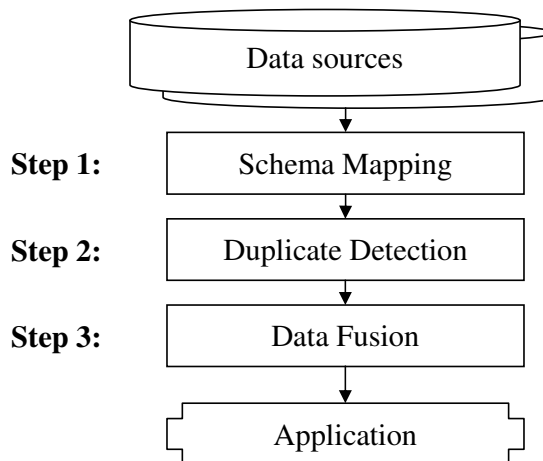


Figure 1: A data integration process

Inconsistencies in data integration. In data integration there are two main kinds of inconsistencies. First, there are *schematic inconsistencies* between data sources; tables not having the same attributes, attributes meaning the same concept but having a different name, or data stored in a different structure. We assume that these inconsistencies already have been resolved in the first step, the step producing a schema mapping. Referring to the small example in Figure 2, a

schema mapping has been established marking the correspondences shown by the solid arrows, for instance identifying that ‘Title’ and ‘Titel’ have the same semantics. The correspondences given by the dotted arrows show the result of *duplicate detection*. This step answers the question of which objects are represented multiple times in the data sources and marks these with the same global id, as shown by the ID column. For instance, we have identified that the first tuples of both relations represent the same real-world movie, ‘Snatch’.

Given these multiple representations of same real world objects, there remain *data inconsistencies*. For instance, the movie ‘Snatch’ is inconsistent in the attribute representing the year the movie was produced. In the remainder of this paper we refer to these data inconsistencies as *data conflicts*, present some strategies on how to get rid of them and, show how these strategies are realized within our research prototype HumMer. The overall goal in this last step is to *fuse* these multiple representations into a single one, thereby resolving the data conflicts.

In a typical data integration setting, there are two types of data conflicts: *contradictions* and *uncertainties*.

Title	Year	Director	Genre	ID
Snatch	2000	Ritchie	Crime	1
Troy	2004	Petersen	History	2
Vanilla Sky	2001	Crowe	Sci-Fi	3
Shrek	2001	Adamson	Anim.	4
The Matrix	1999	Wachowski	Fantasy	5

ID	Titel	Jahr	Rating	Genre
1	Snatch	1999	R	Crime
2	Troja	2004	R	History
3	Vannila Ski	2001	R	Sci-Fi
3	Vannile Sky	2000	16	Comedy
5	Matrix	1999	16	Fantasy

Figure 2: A small example showing two tables with matched attributes and detected duplicates. An uncertainty is shaded with dots; a contradiction is shaded with lines.

Contradictions. A contradiction is a conflict between two or more different NON-NULL values that are all used to describe the same property of an object. In our data integration scenario, this is the case if two or more data sources provide two or more different values for the *same* attribute on the *same* object, *sameness* as given by the correspondences established by *schema matching* and *duplicate detection*. An example of such a contradiction is given in Figure 2, the production year of the movie ‘Snatch’ being 2000 in the left and 1999 in the right data source.

Such contradictions in and among the sources may have several reasons. The most common reason are typographical errors introduced when entering the data, simple misspellings as shown in Figure 2 for the movie with ID 3 in the right data source. Another source of these contradictions is the integration itself, when sources do not agree on the value, e.g., the ‘Year’ of the movie with ID 1. Outdated data values that have not been updated are another reason for contradictions in the data.

Uncertainties. An uncertainty is a conflict between a NON-NULL value and one or more NULL values that are all used to describe the same property of an object. In our scenario, this is caused by missing information, e.g., NULL values in the table, or an attribute completely missing in one table. NULL values present in tables can have different meanings. In general, one distinguishes three different interpretations of NULL values: (i) Value unknown: The value exists, but whoever entered the data did not know it. (ii) Value inapplicable: The corresponding property is not applicable for the object represented by this tuple. (iii) Value withheld: The data exists but we are not allowed to see it.

The reason for considering uncertainties as a special case of conflicts is that it is easier to cope with uncertainties than with contradictions. We deliberately choose to assume most NULL values in a data integration scenario being of type (i). But even with NULL values of type (ii) and (iii) the strategies presented here remain meaningful and do not lead to meaningless results.

Granularity of inconsistencies. Inconsistencies in the data sources can be seen as existing at different granularities, depending on how the real world objects are represented. In the most obvious representation, real world objects are represented in a table, each tuple of the table representing one object whose properties are described by the values in the columns. With this way of modeling the world in mind, contradictions and uncertainties exist at attribute level, the values in the columns being in contradiction or being uncertain. Modeling objects at the granularity of tables and therefore properties being represented by tuples, results in looking at contradicting or uncertain tuples. Modeling properties of objects at the level of tables or even of data sources is possible and results in looking at uncertain/conflicting tables or data sources. Please note that the granularity of the inconsistencies in the real world object does not change, only the granularity of the representation in the data sources. While in Section 3 we consider and handle conflicts at attribute level only, the strategies presented in Section 2 are also valid for other granularities. We want to point out that for the real world objects we want to be able to come up with a fixed, exact and single answer and not with several answers and additional probabilities.

Outline and Contributions. Starting with a description of common conflict handling strategies in Section 2, we define, describe, and analyze in Section 3 conflict handling functions that are used to carry out these different strategies. In the same section we also describe how conflict handling functions are used within our research prototype before presenting related work in Section 4 and concluding in Section 5. We describe and classify different conflict handling strategies and show how they can be realized by using low level conflict handling functions. Properties of these functions are defined and reported upon. We show how the concept of conflict handling functions can be implemented within an integrating information system and used to carry out different strategies in fusing information from multiple heterogeneous sources.

2 Conflict handling strategies

Conflict handling strategies are high level strategies on how to handle inconsistent data. They model an intuition on what to do with inconsistent data. Some of them even describe a decision on what value to take, how to combine values or how to invent a new value, and in that way describe the single, consistent representation created during data fusion.

2.1 Classifying conflict handling strategies

There are several simple strategies to handle inconsistencies, some of which are repeatedly mentioned in the literature [11, 16, 17, 18, 19]. They can be classified as seen in Figure 3, falling into three main classes, two of them consisting of several subclasses. The first division of strategies into the three classes is based on the way they handle (or do not handle) conflicting data: ignorance, avoidance, and resolution.

Conflict ignorance. Conflict ignorance describes strategies that do not make a decision at all. When employing such a strategy one not even needs to be aware of data conflicts in the data, as this information is not needed and not used. These strategies are always applicable and are easy to implement. Two representatives are the PASS IT ON and the CONSIDER ALL POSSIBILITIES strategy:

PASS IT ON. This strategy simply takes all conflicting values, passes them on to the user or another application and lets the user or application decide how to handle possible conflicts.

CONSIDER ALL POSSIBILITIES. This strategy tries to be as complete as possible by enumerating all eventualities and giving the user the choice among all "possible worlds" [6], all possible combinations of attribute values, occasionally creating combinations that are not already present in the sources. Sometimes, as is possible with the MatchJoin operator [22], the user sees only part of all possibilities.

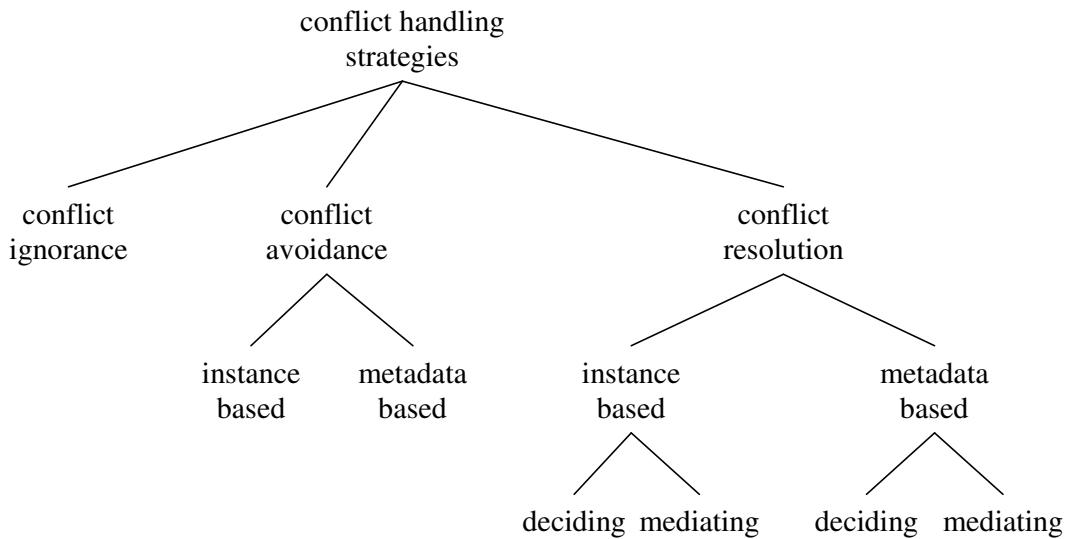


Figure 3: A classification of strategies to handle inconsistent data.

Conflict avoidance. Conflict avoiding strategies do not resolve conflicts, but nevertheless handle inconsistent data. They do not regard the conflicting values before deciding on how to handle inconsistencies. These strategies take a quick decision on whether to handle inconsistencies at all and if yes, which data value to use. Because the decision is often done before regarding the values, these strategies are not always aware of a conflict. Therefore the name of conflict avoidance. They are more efficient from a computational point of view than the conflict resolving strategies considered later on, as the decision can be reached faster. On the other hand they lose precision as not all available information that can be valuable in resolving the conflict is taken into account.

This class can further be divided into two classes, one that takes metadata into account when taking a decision (metadata based) and one that does not (instance based). Two instance based strategies are TAKE THE INFORMATION and NO GOSSIPING:

TAKE THE INFORMATION. The basic idea here is that existing information is taken and information not present is left aside. This strategy leaves aside NULL values and is the natural way of dealing with uncertainties. The concept of subsumption, which filters out unnecessary NULL values and is used in the Minimum Union operator [12], and the use of COALESCE and outer joins in the Merge operator [13] are good examples for the use of this strategy. TAKE THE INFORMATION is only reasonable if there are only uncertainties, but no conflicts in the data.

NO GOSSIPING. If you are unsure how to handle inconsistencies, why not leave them out and report only on the sure facts? This is the strategy used by the consistent query answering approaches [2, 11]. Here, only consistent answers, fulfilling a constraint on the query, are included in the result of a query, leaving aside all inconsistent ones. As the decision is based on the data values and inconsistent answers are ignored this is instance-based conflict avoidance. It is not, as one may think, a conflict ignorance strategy, because it correctly identifies conflicts and is aware of the conflicts.

An example of metadata based conflict avoidance is TRUST YOUR FRIENDS:

TRUST YOUR FRIENDS. The intuition behind this strategy is to trust somebody else to either provide the correct value or the correct strategy. Whom to trust is decided once and carried out for all data values, no matter if there is a conflict or not. This strategy can prefer data from one source over data from other sources and can be observed in the *TSIMMIS* [17] and *Hermes* [19] systems. Intuitively the source preference is given by the user, but this can also be done automatically by choosing the cheapest, most reliable, largest source or using other quality criteria as in the *Fusionplex* system [16].

The main feature in conflict avoidance is that there is first a fundamental decision on whether to handle inconsistencies or not. If they are handled, the decision on what data value to take or not to take is done first, before looking at all the data and returning a result. Therefore, even if there is a conflict in the data, a system using these strategies does not necessarily know about it.

Conflict resolution. In contrast to the previous classes, conflict resolution strategies do regard all the data and metadata before deciding on how to resolve a conflict. This is computationally more expensive than other strategies but provides means to resolve the conflict as optimally as possible.

In contrast to the conflict ignoring and conflict avoiding strategies, conflict resolution strategies can further be subdivided into *deciding* and *mediating* strategies. The main characteristic of a deciding strategy is that it chooses its value from all the already present values. Depending on the class, this choice depends on only the data values or takes metadata into account as well. Mediating strategies on the other hand may choose a value that does not necessarily exist among the conflicting values. They may come up with a new value, that has not existed before.

Therefore, deciding strategies usually enable and allow for data lineage [5], in particular *where-lineage*. In all cases (if ties are broken by additional criteria) it is clear where the value is coming from and therefore can be traced back to its origin. Data lineage information is usually not attached to values created by a mediating strategy as these values can be arbitrary and do not necessarily correspond to one of the existing values. Instance-based, deciding strategies are:

CRY WITH THE WOLVES. The intuition behind this strategy is that correct values prevail over incorrect ones, given enough evidence. It reflects the principle of following the decision of the majority, of choosing the most common value among the conflicting ones. Of course appropriate tie breakers are necessary.

ROLL THE DICE. This strategy considers all values and picks one at random. Although this may not seem to be a very intelligent decision, it is still a valid strategy to resolve conflicts. Lacking any input to decide upon a value, a random value is a good choice. It is still required that one is aware of a conflict, but it has the advantage of being computationally cheap.

An example of a mediating strategy is:

MEET IN THE MIDDLE. This strategy follows the principle of compromise and does not prefer one value over the other but instead tries to invent a value that is as close as possible to all present values. Another principle used can be to minimize the error, or to take the average.

A representative for a deciding strategy based on metadata is:

KEEP UP TO DATE. This strategy uses the most recent value and requires some additional timestamp information about the recency. This information can be present in the tables as a separate attribute or can be provided by other means, such as data lineage facilities. In a data stream environment there is a naturally given order of the tuples coming in so that the recency lies in the data itself.

2.2 Choosing conflict handling strategies.

Choosing a specific conflict handling strategy for a certain problem is not an easy task, when one can choose among several alternatives. In most scenarios, this choice is left to an expert user, as a high level decision, which is then carried out by the integration system. The expert user also needs to specify the strategy in some kind of formalism, e.g., as SQL query to a system. He may also choose a strategy by using a certain system (if this systems only offers one strategy).

The choice of a strategy depends on the domain and is driven by:

- **System availability:** Is there a system available that let me use the strategy? Does the available system restrict the possible choices of strategies?
- **Information availability:** Is there enough information available to choose the strategy? E.g., **TRUST YOUR FRIENDS** and **KEEP UP TO DATE** need some metadata to be used.

- Cost considerations: Do I have enough money/time/disk space to use the strategy? Do I want a cheap result or am I willing to pay for it?
- Quality considerations: Do I want the result to contain as much information as possible or do I just want 'a' result? Quality and cost often depend on each other, a high quality answer often being expensive.

So far, choosing a strategy and reformulating it using some formalism is not automated. There is some room for improvement here, as at least the formulation for the system may be automatized.

3 Conflict Handling Functions

In order to implement conflict handling strategies at attribute level we introduced the concept of conflict resolution functions together with a *data fusion operator* [4]. The main idea of the operator is to group multiple representations of one entity and apply a conflict resolution function to each column and thus fuse the data to a single representation and resolve data inconsistencies.

Conflict resolution can be seen as a more general case of aggregation as known from the standard aggregation functions in SQL. The concept of conflict resolution can be formalized as a function with the conflicting values (and possibly additional parameters) as input and the resolved value as output. Such a function is defined on an input domain, both the domain and the functions itself possess some properties. After defining the concept of conflict resolution, some functions and their properties are presented. As we will see, conflict ignorant and conflict avoiding strategies also fit into the framework and can be modeled as functions.

3.1 A Catalog of functions

There are single- or multi-column functions, each of which can use additional information. An n -ary single column conflict handling function is a function f_{ch} defined on a domain D and maps n input values to one output value of the same or another domain S . Conflicting values (c_i) are resolved to a solution s :

$$f_{ch} : D^n \mapsto S \tag{1}$$

$$f_{ch}(c_1, \dots, c_n) = s, s \in S, c_i \in D, \forall i = 1 \dots n$$

An n -ary multi-column conflict handling function is a function f_{ch} defined on m domains D_j and maps n input m -tuples to one output value from another domain. The idea here is that conflicts are resolved in a column by using additional knowledge from other columns as well. The correspondences between values from the different columns are not lost, therefore the m -tuples:

$$f_{ch} : D_1^n \times \dots \times D_m^n \mapsto S \tag{2}$$

$$f_{ch}((c_1^1, \dots, c_1^m), (c_2^1, \dots, c_2^m), \dots, (c_n^1, \dots, c_n^m)) = s,$$

$$s \in S, c_i^j \in D_j, \forall i = 1 \dots n, \forall j = 1 \dots m$$

For a single column function (as shown below) additional information is given as a separate parameter A . Additional information for multi-column functions is analogous.

$$f_{ch} : D^n \times A \mapsto S \tag{3}$$

$$f_{ch}(c_1, \dots, c_n, a) = s, a \in A, s \in S, c_i \in D, \forall i = 1 \dots n$$

The functions are defined with a fixed number of n , the number of conflicting values. In the case of n being 1 there is no conflict and every function is usually evaluated as $f_{ch}(c_1) = c_1$. In the general case, when we use the functions as part of a real system we may not know the exact number of conflicting values: n may vary. The general goal is to use these functions to implement different strategies from Section 2. Domains of input and output of conflict resolutions functions can be one of four types:

Function	Description
COUNT	Counts the number of distinct NON-NULL values, i.e., the number of conflicting values. Only <i>indicates</i> conflicts, the actual data values are lost.
MIN / MAX	Returns the minimal/maximal input value with its obvious meaning for numerical data. Lexicographical (or other) order is needed for non numerical data.
SUM / AVG / MEDIAN	Computes sum, average, and median of all present NON-NULL data values.
VARIANCE / STDDEV	Returns variance and standard deviation of data values.
RANDOM	Randomly chooses one data value among all NON-NULL data values.
CHOOSE	Returns the value supplied by a specific source.
COALESCE	Returns the first NON-NULL value appearing.
FIRST / LAST	Returns the first/last value, even if it is a NULL value.
VOTE	Returns the value that appears most often among the present values. Ties can be broken by a variety of strategies, e.g., choosing randomly.
GROUP	Returns a set of all conflicting values. Leaves conflict resolution to the user.
SHORTEST / LONGEST	Chooses the value of minimum/maximum length according to a length measure.
(ANNOTATED) CONCAT	Returns the concatenated values. May include annotations, such as the names of the data sources.
HIGHEST QUALITY	Returns the value of highest information quality. Requires an underlying quality model.
MOST RECENT	Returns the most recent value. Recency is evaluated with the help of another attribute or other metadata about tuples/values.
MOST ACTIVE	Returns the most often accessed or used value. Access statistics of the DBMS can be used in evaluating this function.
CHOOSE DEPENDING	Chooses the value v in column A that belongs to a specific given value c in another column B . B and c are given.
CHOOSE CORRESPONDING	Chooses the value v in column A that belongs to the value v' already chosen for another column B . B is given.
MOST COMPLETE	Returns the value of the source that contains the fewest NULL values in the attribute in question.
MOST DISTINGUISHING	Returns the value that is the most distinguishing among all present values in that column.
MOST GENERAL CONCEPT / MOST SPECIFIC CONCEPT	Using a taxonomy or ontology this function returns the more general value (lowest common ancestor) or the more specific value (if the values are on a common path in the taxonomy).

Table 1: Conflict handling functions from [4], which can be used to implement conflict handling strategies.

- **Numerical (N)**: Numerical domains consist of numbers, the domain is ordered and infinite (e.g., integers: 1, 2, 3, ...).
- **Strings (S)**: The String domain consists of words, based on characters. There exists an order (lexicographical order) and the number of values is infinite (e.g., names: Connery, Hanks, Cage, ...)
- **Categorical (C)**: There is no order defined on the values of the domain and the number of elements is finite. The elements can be strings or numbers (e.g., colors: {red, green, blue}, or grades: {1, 2, 3, 4, 5}).
- **Taxonomical (T)**: A taxonomical domain consists of entities with a semi-order defined on them. The entities in the domain can be seen as structured like a tree and there is a finite number of elements (e.g., locations: Berlin, Germany, Europe, World, ...).

As we show later, some of the functions require input data of a certain domain-type whereas other functions can be used with input data of more or all domains. Table 1 describes some possible functions (taken from [4]). The properties of these functions are further examined in the next section.

3.2 Properties of conflict handling functions

In order to understand conflict resolution functions better, we present and discuss some of their properties. Some of the properties are described in more detail in [7] and other sources, some of them become important in the context of our research system, when considering the efficient

implementation of the functions, the optimization of query plans involving such functions and when learning functions from user input to make suggestions what functions should be used on a column. Table 2 gives an overview of the functions and their properties:

- **Type:** A function is either a single- or multi-column function, depending on how many columns are used to decide on the resolved value. In addition, there can be other data used, too (parameter function) or not be used (parameter free function). All standard aggregation functions known from SQL are examples for single column, parameter free functions, whereas CHOOSE DEPENDING is an example of a multi-column parameter function.
- **Applicable input domains:** For actually computing a conflict resolution it is important to know the different domains a function can work on. Functions like SUM or AVERAGE are defined on a numerical domain only, whereas MOST GENERAL CONCEPT requires a taxonomic input domain.
- **Mediating:** Similar to the concept of mediating strategies, a function is mediating, if $f_{ch}(c_1, \dots, c_n) = y$ meaning that a new value is created (e.g., AVERAGE, CONCAT). This does not allow for *where-lineage*.
- **Deciding:** Deciding functions choose among the already present values, e.g., (MAX or SHORTEST) where $f_{ch}(c_1, \dots, c_i, \dots, c_n) = c_i, i \in \{1, \dots, n\}$. We assume that ties (e.g., two shortest values) are broken by a secondary criterion, e.g., the order of the values, to always get a defined result. This way it is possible to assign *where-lineage* to the value.
- **Monotonicity:** Monotonicity is defined on numerical domains as $\forall i = 1 \dots n, x_i \neq \perp, y_i \neq \perp, x_i \leq y_i \Rightarrow f_{ch}(x_1, \dots, x_n) \leq f_{ch}(y_1, \dots, y_n)$. This property is only applicable when an order is defined on the domain, so only for numerical and string domains. MIN and MAX are examples for monotone functions as are FIRST and CONCAT.
- **Idempotency:** Idempotency ensures that functions can also cope with non-conflicting data, as $f_{ch}(c_1, \dots, c_1)$ is always evaluated to c_1 . Most functions are idempotent, exceptions include COUNT and CONCAT.
- **Boundary condition:** This property ensures that minimal/maximal values are always resolved to itself, i.e. $f_{ch}(\perp, \dots, \perp) = \perp$ and $f_{ch}(\top, \dots, \top) = \top$ with \perp/\top being the lower/upper boundary of the domain. It is a special case of idempotency defined on the borders of the input domain, used to define aggregation functions in [7].
- **Symmetry, or order sensitivity:** Parameters of a commutative binary function can be exchanged without influencing the result i.e. $f_{ch}(c_1, c_2) = f_{ch}(c_2, c_1)$. Symmetry is an extension to n -ary functions and shows if the function is sensitive to the order of the input values. A function is symmetric, if the order of the conflicting values does not change the result, i.e. if $f_{ch}(c_1, c_2, c_3) = f_{ch}(c_1, c_3, c_2) = \dots = f_{ch}(c_3, c_2, c_1)$. COALESCE and FIRST are examples for non symmetric functions whereas CHOOSE or VOTE are among the symmetric functions.
- **Duplicate sensitivity:** Duplicate sensitivity indicates that the result is influenced by duplicate input values, i.e., $f_{ch}(c_1, c_1, c_2) \neq f_{ch}(c_1, c_2)$. COUNT, SUM, and VOTE are examples of duplicate sensitive functions, whereas MIN, LONGEST and MOST GENERAL are examples for insensitive ones.
- **Associativity, or decomposability:** Associativity extended to n -ary functions allows for the computation of partial results and their combination without changing the overall result. It is defined as $\forall n, m \in \mathbb{N}, \forall x_1, \dots, x_n, y_1, \dots, y_m : f_{ch}(x_1, \dots, x_n, y_1, \dots, y_m) = f_{ch}(f_{ch}(x_1, \dots, x_n), f_{ch}(y_1, \dots, y_m))$. Many functions are associative, exceptions are e.g. RANDOM and VOTE.
- **Neutral element:** A neutral element, if one exists, is a value from the input domain that can be omitted, because it has no impact on the result of the function, $f_{ch}(c_1, \dots, c_n) = f_{ch}(c_1, \dots, c_{i-1}, c_{i+1}, \dots, c_n)$ with neutral element c_i . The neutral element of, e.g., SUM is 0, for COALESCE it is \perp .

Function	type ^a	applicable input domains ^b	mediating, deciding	monotonicity	boundary condition	idempotency	symmetry, order sensitivity	duplicate sensitivity	associativity (decomposability)	neutral element	annihilator	based on ^c
COUNT	S	A	M	+	-	-	+	+	-	-	-	D
MIN / MAX	S	S,N	D	+	+	+	+ ^e	-	+	+ ^d	+ ^d	D
SUM	S	N	M	+	-	-	+	+	+	+	-	D
AVG / MEDIAN	S	N	M	+	+	+	+	+	-	-	-	D
VARIANCE / STDDEV	S	N	M	-	-	-	+	+	-	-	-	D
RANDOM	S	A	D	-	+	+	-	+	-	-	-	D
CHOOSE	SP	A	D	-	+	+	+	+	+	-	-	MD
COALESCE	S	A	D	-	+	+	-	+	+	+	-	D
FIRST / LAST	S	A	D	+	+	+	-	+	+	-	-	D
VOTE	S	A	D	-	+	+	+ ^e	+	-	-	-	D
GROUP	S	A	×	×	×	×	×	×	×	×	×	D
SHORTEST / LONGEST	S	S,C,T	D	-	+	+	+ ^e	-	+	+ ^d	+ ^d	D
(ANNOTATED) CONCAT	S	A	M	+	-	-	-	+	+ ^f	+ ^f	-	D
HIGHEST QUALITY	SP	A	D	-	+	+	+	+	+	-	-	MD
MOST RECENT	SMP	A	D	-	+	+	+	+	+	-	-	MD
MOST ACTIVE	SP	A	D	-	+	+	+	+	+	-	-	MD
CHOOSE DEPENDING	MP	A	D	-	+	+	+	+	+	-	-	D
CHOOSE CORRESPONDING ^g	MP	A	D	-	+	+	+	+	+	-	-	MD
MOST COMPLETE	SP	A	D	-	+	+	+	+	+	-	-	MD
MOST DISTINGUISHING	SP	A	D	-	+	+	+	+	+	-	-	MD
MOST GENERAL/SPECIFIC CONCEPT	SP	T	M,D	-	×	+	+	-	+	-	+	MD

^a(S)ingle column, (M)ultiple column, with (P)arameter

^bN(umeric), S(tring), C(ategorical), T(axonomical) or A(II)

^cfunction uses only data (D) or also metadata (MD) in computing a result

^donly if *top* and *bottom* elements exist

^edepending on tie breaker

^fnot the annotated version

^gOnly in combination with a deciding function and in order to decide upon the properties, the properties of the other function also plays a role.

Table 2: Conflict handling functions and some of their properties, × meaning *not applicable*, a plus (+) marking *has the property* and a minus (-) *has not the property*.

- **Annihilator:** An annihilator a is a value that, included in the input, determines the result, i.e., $f_{ch}(c_1, \dots, c_i, \dots, c_n) = c_i$ with annihilator c_i . c_i is also called the veto, or absorbing element. Annihilators exist with MAX/MIN (maximal/minimal value ever) and MOST GENERAL CONCEPT (root element of taxonomy).

The next sections show how these functions realize some of the strategies already mentioned and how they are used within our research system.

3.3 Realizing conflict handling strategies

Some strategies from Section 2 have a direct equivalent among the functions and can easily be realized by just applying this function to conflicting data. First to mention is the simple conflict ignoring strategy PASS IT ON which is very easily carried out by the functions GROUP or CONCAT, the former requiring for a special set data type. As already mentioned in Section 2, the COALESCE function can be used to implement the TAKE THE INFORMATION strategy. TRUST YOUR FRIENDS is best illustrated by the CHOOSE function, as source preference. This is also a good example that there may be different ways of realizing a strategy. Source preference can also be accomplished by

Strategy	Possible functions to realize the strategy
PASS IT ON	GROUP, CONCAT
CONSIDER ALL POSSIBILITIES	[6, 22]
TAKE THE INFORMATION	COALESCE, LONGEST
NO GOSSIPING	[2, 11]
TRUST YOUR FRIENDS	CHOOSE, CHOOSE DEPENDING, HIGHEST QUALITY, FIRST, MOST COMPLETE, CHOOSE CORRESPONDING
CRY WITH THE WOLVES	VOTE
ROLL THE DICE	RANDOM
MEET IN THE MIDDLE	AVERAGE, MEDIAN, MOST GENERAL
KEEP UP TO DATE	MOST RECENT, FIRST

Table 3: Strategies and functions that can be used to realize them.

using CHOOSE DEPENDING with a column that contains the source name and the desired source name as second parameter, or with HIGHEST QUALITY and a quality measure as parameter that favors the desired source. Our further findings are summarized in Table 3.

3.4 Conflict handling in an integrated information system

We are currently developing an integrated information system, namely the HumMer¹ (short for Humboldt-Merger) system [3]. The system virtually queries and fuses data from distributed, heterogeneous, relational data sources. In our system we assume a three step information integration process as presented in Section 1, where after resolving schematic conflicts and finding duplicate objects, only data conflicts remain.

Potential users can interact with the system in two different ways. First, one can use a wizard that interactively guides through the entire integration process, and gives the opportunity to influence how schemata are matched and how duplicates are detected. At the end users are able to specify conflict handling at attribute level. Second, users and applications have the opportunity of formulating FUSEBY queries (see [4] for syntax and semantics) and pose it to the system. A FUSEBY query allows the specification of attribute level conflict resolution in a single SQL like statement. Using this interaction mode, schema matching and duplicate detection are triggered automatically, using default parameters.

In both ways of interacting with the system, conflict handling is specified on an attribute level by choosing one conflict handling function per column. Figure 4 shows the last step in the integration wizard where a user can define conflict handling functions. In the Figure, you see that the tuples are already grouped by their corresponding real world entities as determined by duplicate detection, marked in Figure 4. Also, function MAX has been selected for the *Age* column and function VOTE for the *Student* column.

3.5 Implementing conflict handling functions

Conflict handling functions in HumMer are implemented as extended aggregation functions, operate at attribute level and can handle different numbers of conflicting values (e.g. two for Bob, three for Alice and only one for Peter, see Figure 4). The system stores and uses some function metadata like in- and output domain type, and information on some properties (currently only *order sensitivity* and *function type*). The functions are used in a fusion operator, which first groups tuples and then applies the functions to all values from each group.

There are two different application modes, called offline and online mode. In offline mode, all conflicting values are given to the function at once and then the result is computed, whereas in online mode, the values are processed one after the other, at each step providing a partial result of conflict handling considering the values so far. The online version allows for functions, that do not need to process all input values, to stop as soon as they have computed their result, resulting in a slightly faster runtime whereas with the offline version the system needs to regard all values, when they are passed to the function. We implemented both versions for most of the functions in HumMer although some (e.g. FIRST, COALESCE) are predestined to be used in online mode.

¹<http://www.informatik.hu-berlin.de/mac/hummer/>

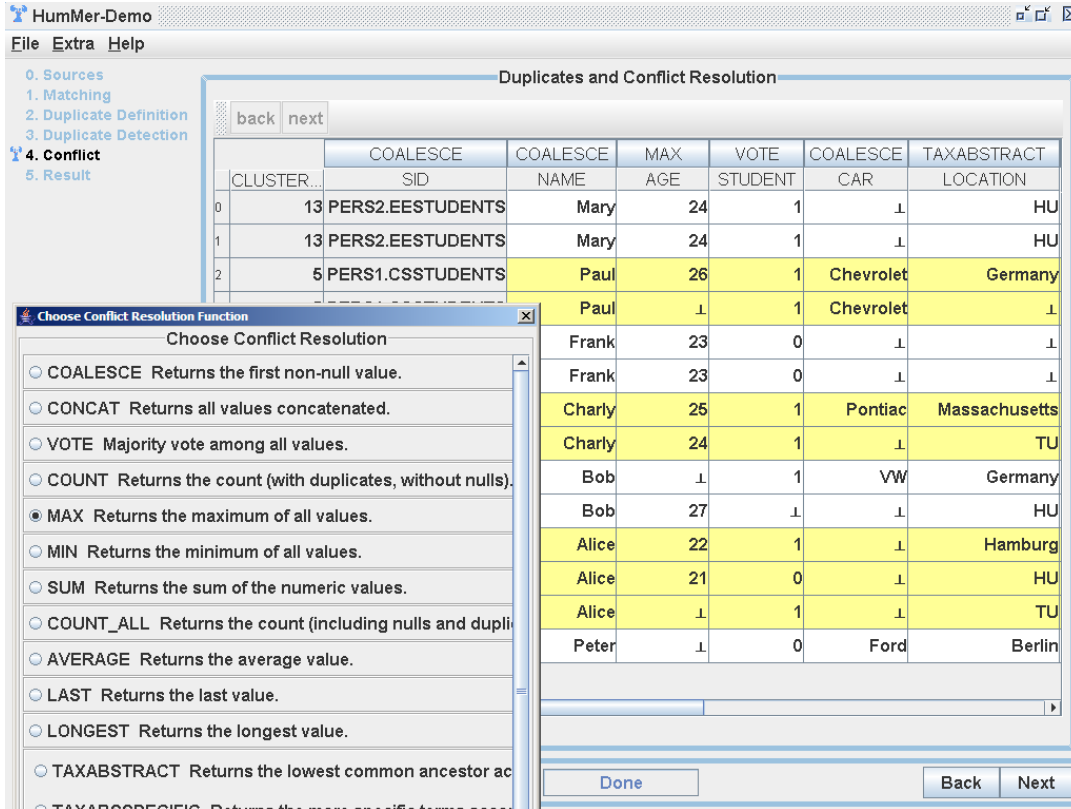


Figure 4: Screenshot of the integration wizard (data fusion step), where a user can specify conflict handling at attribute level.

Additional parameters (e.g., a taxonomy for the taxonomic functions) are given to the function before the values are processed, multi-column functions are realized by passing the values from all necessary columns to the function. Conflict handling functions in HumMer build on the concept of metadata functions from XXL [10] and are extensible in the sense that new functions can be easily incorporated in the system by adding new functions.

4 Related Work

Apart from the systems already mentioned in the classification of the strategies in Section 2, we briefly give an overview on related work. There are many integrating information systems that provide a unified view to multiple heterogeneous sources. However, the problem of data conflicts—first identified by Dayal [8]—and how to resolve them, is mentioned or tackled by only a few. Among them are *TSIMMIS* [17] and *Fusionplex* [16], which both use the TRUST YOUR FRIEND strategy by choosing data from a specific source, source preference either given by the user or by some data quality criteria. The *Hermes* system [19] explicitly mentions and is able to use five different strategies based on data and metadata. *ConQuer* [11] is a system that handles inconsistencies by filtering them out and therefore realizes a NO GOSSIPING strategy. A similar approach using certain answers instead of consistent answer is used in the INFOMIX system [14]. In the context of data warehouses, Burdick et. al. enhance on a the simple PASS IT ON strategy by using combinations of all present attribute values to return all possible answers to a query [6]. Probabilistic approaches ([9, 15, 21]) follow on a similar path by enhancing attribute values with a probability of being an answer. The *FraQL* system [18] uses a predefined set of user defined aggregation functions (some of them with two parameters) to realize some strategies. This dovetails with the work of Wang and Zaniolo [20] who show how to implement arbitrary single column aggregation functions. Other relational approaches, such as standard or more advanced relational

operators (*join*, *minimum union* [12], *match join* [22] or *merge* [13]), are only able to handle uncertainties but not contradictions. There are results on aggregation functions from the fuzzy systems community [7]. Aggregation functions are defined as single column functions, defined on the interval $[0, 1]$, and fulfilling a boundary and monotonicity condition. Our functions extend this notion of aggregation functions. A special case of data fusion is the resynchronization of replicated databases where conflicts between attribute values of two different database versions are resolved using some of the strategies mentioned before [1]. In summary, all approaches so far are somehow limited to a few predefined strategies or functions and do not allow for a flexible on-demand specification of conflict resolution as in the HumMer system.

5 Conclusions and future work

We consider a three step data integration process and are mostly concerned with the third step of *data fusion*, where multiple representations of the same real world entity are fused to a single representation. In this step the difficulty lies in handling the conflicts at data level. We described and classified different conflict handling strategies and show how they can be realized by using low level conflict handling functions. Properties of these functions have been defined and reported upon. We showed how the concept of conflict handling functions is implemented into our integrating information system HumMer and how they are used to carry out different strategies in fusing information from multiple heterogeneous sources.

Future work includes the further investigation of relevant and interesting properties, such as order-/duplicate sensitivity and associativity, of both strategies and functions, the classification of functions as well as more robust and efficient implementations of already existing or completely new conflict handling functions in the HumMer system.

Furthermore, the knowledge on the properties of the functions will enable the optimization of queries posed in the HumMer context, that do not only contain conflict handling functions but also other relational operators as well. The definition of transformation rules for the data fusion operator and such functions will be a next step in this line of research. We are also working on the automatic recommendation of conflict handling functions based on past user decisions on conflict handling.

Acknowledgment. This research was supported by the German Research Society (DFG grant no. NA 432).

References

- [1] Oracle Database Advanced Replication, 10G Release 2, Chapter 5, Conflict Resolution Concepts and Architecture.
- [2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS*, pages 68–79, Philadelphia, PA, 1999.
- [3] A. Bilke, J. Bleiholder, C. Böhm, K. Draba, F. Naumann, and M. Weis. Automatic data fusion with hummer. In *Proc. of VLDB*, pages 1251–1254, Trondheim, Norway, 2005.
- [4] J. Bleiholder and F. Naumann. Declarative data fusion - syntax, semantics, and implementation. In *Proc. of ADBIS*, pages 58–73, Tallinn, Estonia, 2005.
- [5] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Proc. of ICDT*, pages 316–330, London, UK, 2001.
- [6] D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Olap over uncertain and imprecise data. In *Proc. of VLDB*, pages 970–981, Trondheim, Norway, 2005.
- [7] T. Calvo, G. Mayor, and R. Mesiar, editors. *Aggregation Operators - New Trends and Applications*. Physica-Verlag, Heidelberg, 2002.
- [8] U. Dayal. Processing queries over generalization hierarchies in a multidatabase system. In *Proc. of VLDB*, pages 342–353, Florence, Italy, 1983.

- [9] L. G. DeMichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Trans. Knowl. Data Eng.*, 1(4):485–493, 1989.
- [10] J. V. den Bercken, B. Blohsfeld, J.-P. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL - a library approach to supporting efficient implementations of advanced database queries. In *Proc. of VLDB*, pages 39–48, Rome, Italy, 2001.
- [11] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *Proc. of SIGMOD*, pages 155–166, Baltimore, MD, 2005.
- [12] C. A. Galindo-Legaria. Outerjoins as disjunctions. In *Proc. of SIGMOD*, pages 348–358, Minneapolis, MN, 1994.
- [13] S. Greco, L. Pontieri, and E. Zumpano. Integrating and managing conflicting data. In *Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 349–362, 2001.
- [14] N. Leone, G. Greco, G. Ianni, V. Lio, G. Terracina, T. Eiter, W. Faber, M. Fink, G. Gottlob, R. Rosati, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszki. The infomix system for advanced integration of incomplete and inconsistent data. In *SIGMOD Conference*, pages 915–917, 2005.
- [15] E.-P. Lim, J. Srivastava, and S. Shekhar. Resolving attribute incompatibility in database integration: An evidential reasoning approach. In *Proceedings of the Tenth International Conference on Data Engineering, February 14-18, 1994, Houston, Texas, USA*, pages 154–163. IEEE Computer Society, 1994.
- [16] A. Motro, P. Anokhin, and A. C. Acar. Utility-based resolution of data inconsistencies. In *Proc. of IQIS Workshop*, pages 35–43, Paris, France, 2004.
- [17] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc. of VLDB*, pages 413–424, Bombay, India, 1996.
- [18] E. Schallehn, K.-U. Sattler, and G. Saake. Efficient similarity-based operations for data integration. *Data and Knowledge Engineering*, 48(3):361–387, 2004.
- [19] V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward. Hermes: A heterogeneous reasoning and mediator system. Technical report, University of Maryland, 1995.
- [20] H. Wang and C. Zaniolo. Using SQL to build new aggregates and extenders for object-relational systems. In *Proc. of VLDB*, pages 166–175, Cairo, Egypt, 2000.
- [21] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.
- [22] L. L. Yan and M. T. Özsu. Conflict tolerant queries in AURORA. In *Proc. of CoopIS*, page 279, Edinburgh, UK, 1999.