Conflict Handling Strategies in an Integrated Information System

Jens Bleiholder and Felix Naumann
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 within and among the sources. We present a classification of conflict resolution strategies and show how these are implemented within an integrated information system, the Humboldt-Merger.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases

General Terms

Algorithms, Design, Languages, Theory

Keywords

data integration, data fusion, conflict handling, conflict resolution

1. DATA FUSION

Integrated (relational) information systems provide users with a unified view of heterogeneous data sources. The tasks of querying the underlying data sources, combining the results, and presenting them to the user are 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*, which can be 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. *Duplicate detection* techniques find multiple, possibly inconsistent representations of same real world objects. Third, as a last step, the duplicate representations can be combined and fused together into a single representation while inconsistencies in the data are resolved. This last step is referred to as *data fusion* and is the main focus in this paper.

WWW2006, May 22–26, 2006, Edinburgh, UK.

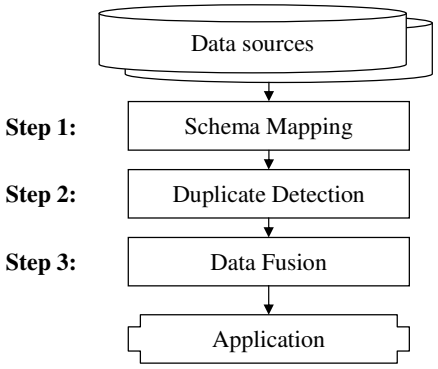
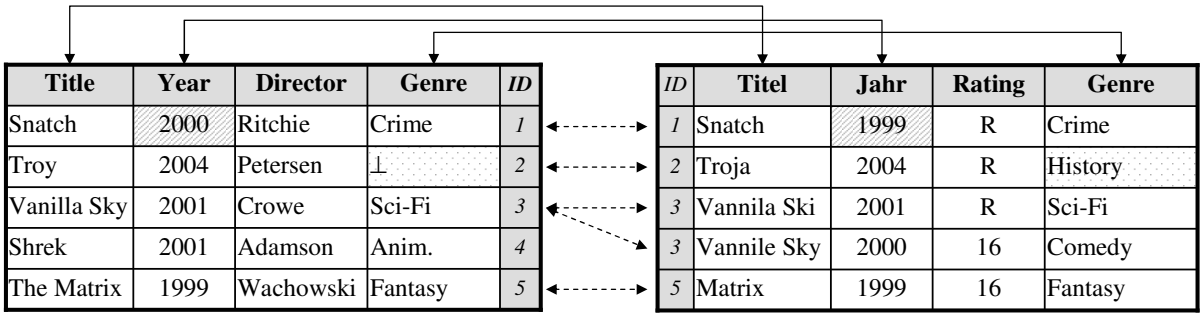


Figure 1: A data integration process

Inconsistencies and 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 schema mapping step. 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 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 a common global id, as shown by the ID column. For instance, we have identified that the first tuple of both relations represents the same real-world movie, namely '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* and present some strategies on how to handle them. We also show how these strategies are realized within our research prototype—the Humboldt-Merger or HumMer. The overall goal in this last step is to *fuse* these multiple representations into a single one, while resolving any data conflicts.



Title	Year	Director	Genre	ID
Snatch	2000	Ritchie	Crime	1
Troy	2004	Petersen	⊥	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.

Data conflicts. In a typical data fusion setting, there are two types of data conflicts: *contradictions* and *uncertainties*. 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, *same*ness as given by the correspondences established by *schema mapping* 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-hand and *1999* in the right-hand data source.

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 either caused by missing information, e.g., NULL values in the table, or caused by an attribute completely missing in one table. In the latter case, we assume that the missing attribute values are padded with NULL values.

Outline and contributions. First, we describe and classify common conflict handling strategies in Section 2. Then, in the same section we describe how conflict handling functions are used within our research prototype to implement those strategies before presenting related work in Section 3 and concluding in Section 4. A more detailed definition, description, and analysis of conflict handling functions that are used to carry out different strategies can be found in [4]. Further details on both the general setting and the role of functions, together with a list of function properties, can also be found in [4]. In this introductory paper we describe and classify different conflict handling strategies and show how they can be realized by using low level conflict handling functions within an integrated information system that fuses 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. In this section we classify these strategies and, show how these are implemented as functions and finally lay out the actual implementation in the HumMer prototype.

2.1 Classifying conflict handling strategies

There are several simple strategies to handle inconsistencies, some of which are repeatedly mentioned in the literature [10, 13, 15, 16, 17]. They can be classified as seen in Figure 3 and fall into three main classes. 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.

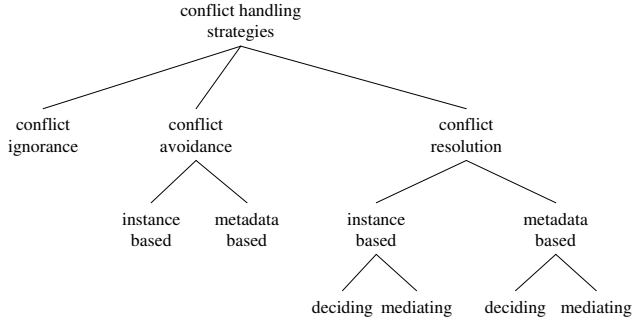


Figure 3: A classification of strategies to handle inconsistent data.

Conflict ignorance. Conflict ignorance describes strategies that do not make a decision with respect to conflicts at all. When employing such a strategy the system does not even need to be aware of conflicts in the data, as this information is not needed and not used. These strategies are feasible in any integration situation 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 among the values.

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, occasionlly creating combinations that are not already present in the sources. Sometimes, as is possible with the MatchJoin operator [19], the user sees only part of all possibilities.

Conflict avoidance. Conflict avoiding strategies do not actually resolve conflicts, but they 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 made before regarding the data values, or without looking at all data values, these strategies are not always aware of each individual 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 [11], and the use of COALESCE and outer joins in the Merge operator [12] are good examples for the use of this strategy. TAKE THE INFORMATION is 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 certain facts? This is the strategy used by the consistent query answering approaches [1, 10]. Here, only consistent answers, fulfilling certain constraints on the query, are included in the result of a query leaving aside all inconsistent ones. Because the decision is based on the data values and inconsistent answers are ignored, this strategy reflects instance-based conflict avoidance. It is not, as one may think, a conflict ignorance strategy, because it correctly separates conflicting from non conflicting data.

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

TRUST YOUR FRIENDS. The intuition behind this strategy is to trust a third party 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* [15] and *Hermes* [17] 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 by using other quality criteria as in the *Fusionplex* system [13].

The main feature in conflict avoidance is that there is an initial 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.

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 approach is computationally more expensive than other strategies but provides means to resolve the conflict as flexibly as possible.

In contrast to the conflict ignoring and conflict avoiding strategies, conflict resolution strategies are further 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.

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 of 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 conflicting 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 inexpensive.

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 time-stamp 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.

Function	Description
COUNT	Counts the number of distinct NON-NULL values, i.e., the number of conflicting values. Only *indicates* 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[3]) to implement conflict handling strategies.

Strategy	Possible functions to realize the strategy
PASS IT ON	GROUP, CONCAT
CONSIDER ALL POSSIBILITIES	[6, 19]
TAKE THE INFORMATION	COALESCE, LONGEST
NO GOSSIPING	[1, 10]
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 2: Strategies and functions that can be used to realize them.

2.2 Implementing conflict handling strategies

Conflict handling *strategies* are carried out by using conflict handling *functions* in a specific integration framework. Table 1 shows some possible functions; properties of these functions are defined and discussed in [4]. Some of the strategies from Section 2 have a direct equivalent among these 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 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 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 2.

2.3 Conflict handling in the HumMer system

We are currently developing the integrated information system HumMer[1] (short for Humboldt-Merger) [2]. The system performs virtual integration in querying and fusing 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

[1] `http://www.informatik.hu-berlin.de/mac/hummer/`

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 [3] for syntax and semantics) and pose them 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, during which users can specify conflict handling functions. In the background are original tuples already grouped as determined by duplicate detection. Also, function MAX has been selected for the *Age* column and function VOTE for the *Student* column.

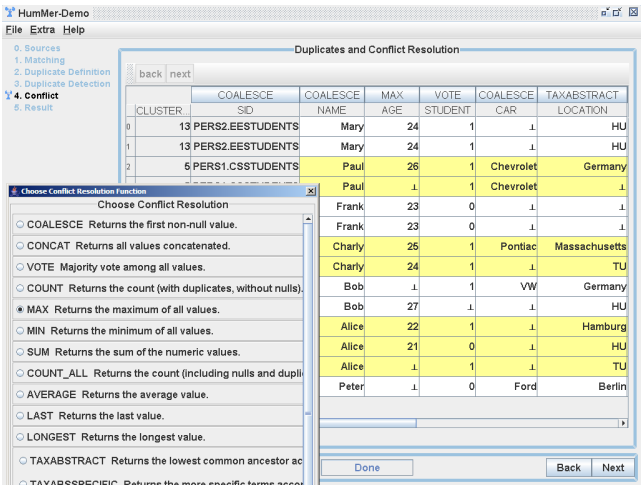


Figure 4: Screenshot of the integration wizard (data fusion step), where a user can specify conflict handling functions at attribute level.

2.4 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.

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 [9] and are extensible in the sense that new functions can be easily incorporated.

3. 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 integrated 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* [15] and *Fusionplex* [13], 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 [17] explicitly mentions and is able to use five different strategies based on data and metadata. *ConQuer* [10] is a system that handles inconsistencies by filtering them out and therefore realizes a NO GOSSIPING strategy. The *FraQL* system [16] uses a predefined set of user defined aggregation functions (some of them with two parameters) to realize some of the strategies mentioned in this paper. This dovetails with the work of Wang and Zaniolo [18] who show how to implement arbitrary single column aggregation functions.

Purely relational approaches, such as standard or more advanced relational operators (*join*, *minimum union* [11], *match join* [19] or *merge* [12]), are only able to handle uncertainties but not contradictions. There are results on aggregation functions from the fuzzy systems community [7]. There, aggregation functions are defined as single column functions on the interval $[0, 1]$, 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 [14].

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.

4. 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 high-level conflict handling strategies and show how they are realized using low-level functions in our integrated information system HumMer.

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, 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 algebraic reformulation 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 resolution.

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

5. REFERENCES

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS*, pages 68–79, Philadelphia, PA, 1999.

[2] 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. Demonstration.

[3] J. Bleiholder and F. Naumann. Declarative data fusion - syntax, semantics, and implementation. In *Proc. of ADBIS*, pages 58–73, Tallinn, Estonia, 2005.

[4] J. Bleiholder and F. Naumann. Conflict handling strategies in an integrated information system. Technical Report Informatik-Berichte, Nr. 197, Humboldt-Universität zu Berlin, 2006.

[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] 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.

[10] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *Proc. of SIGMOD*, pages 155–166, Baltimore, MD, 2005.

[11] C. A. Galindo-Legaria. Outerjoins as disjunctions. In *Proc. of SIGMOD*, pages 348–358, Minneapolis, MN, 1994.

[12] 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.

[13] 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.

[14] Oracle Database Advanced Replication, 10G Release 2, Chapter 5, Conflict Resolution Concepts and Architecture.

[15] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc. of VLDB*, pages 413–424, Bombay, India, 1996.

[16] E. Schallehn, K.-U. Sattler, and G. Saake. Efficient similarity-based operations for data integration. *Data and Knowledge Engineering*, 48(3):361–387, 2004.

[17] 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.

[18] 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.

[19] L. L. Yan and M. T. Özsu. Conflict tolerant queries in AURORA. In *Proc. of CoopIS*, page 279, Edinburgh, UK, 1999.