Integrating XML Databases

Dennis Schroer
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
g.h.schroer@student.utwente.nl

ABSTRACT
Uncertainty generated during a data integration process is often used to avoid information loss. This uncertainty could exist in the resulting data itself, but also in the generated mappings between databases. XML databases provide a flexible but structured way of saving data, which suits uncertain data well. This paper describes a way to integrate data. Several problems arise here which involve one-to-many or many-to-many mappings or relations and different domains. The assignment of probabilities is discussed, as well as a way to represent uncertainty in XML. We validate these methods by conducting an experiment that integrates data from two publication repositories of the University of Twente: the EPrints repository of the Faculty of EEMCS and the university-wide Metis repository. The result is compared with an ordinary integration method without uncertainty.

Keywords
Database Merging, Database Integration

1. INTRODUCTION
In many situations merging of heterogeneous databases into a single database is needed. By integrating data from different sources, one can acquire a more complete set of information. As example, think of combining two customer lists in order to create one list containing all your customers after your company merged into another company.

In most cases however, the heterogeneous databases have different schemata. In order to integrate the data, a mapping between the fields in the various databases is created. This mapping however could contain one-to-many or even many-to-many mappings. Objects in a database could be related to multiple objects in a one-to-many or many-to-many relation, like a publication having multiple authors. Furthermore, the data in both databases about the same real world objects (RWO) could differ or be not complete.

Example 1 As example, think of two databases A and B, where database A contains tuples (name, address) and database B contains tuples (name, street, number). Database A contains a record ("John", "Mainstreet 1") and database B contains a record ("John", "Mainstreet", 3). It is not sure if both records contain the address of the same person. Next to that, in order to compare the addresses, a translation between the fields is needed.

The merging and integration of data has been studied before. A small part of the research conducted during last decades involves uncertain data. Uncertainty in data means for instance that the two addresses in example 1 are both saved with probability 0.5, which means the one person John live at either Mainstreet 1 or Mainstreet 3. Moreover, it could even be the case that there exists two persons with the name John. Uncertainty could thus also be involved in the decision whether two records store information about the same RWO, or even in the mapping between the databases. This leads to a lot of uncertainty and a description of methods to merge and save data along with their probabilities is needed.

1.1 Problem Statement
Most researches however focus on the creation of the mappings or the notation or representation of uncertainty. Few of the researches however focus on the integration of fields with one-to-many relations, many-to-many relations or fields of which the domains differ.

The integration of the data consists of two sub problems. Firstly a method is needed to recognize if two records contain data about the same real world object. The result of this method is a probability ranging from 'no match' to 'match'.

Secondly a method is needed to actually integrate the data. Each field has to be merged with the corresponding field or fields in the other record. All possibilities get assigned probabilities. A calculation is needed to assign these probabilities.

A difficulty in this method is merging data which is related via a one-to-many or many-to-many mappings. Additional mappings are needed here. In example 1, address in database A would be mapped with street and number in database B, where street is followed by a space and the number.

Besides these mappings, there are also one-to-many and many-to-many relations. A published paper might have multiple authors, but what if one database says John was one of the authors, while the other states he is not one of the authors? This causes a problem which needs to be solved.

Another difficulty is merging two fields with different domains. Some data, like house number, could be saved as text or as a number. Other domain differences include maximum string length and default values. A translation is required here.

As a result of the above methods, a lot of uncertainty is introduced. In order to preserve this, a structured way to store this in XML is needed. Low probabilities should be filtered out, as simply storing all possible combinations would require too much storage space [3].

Based on the problems stated above, the following research questions have been composed:

1. How can data in one-to-many or many-to-many mappings and relations be integrated?
2. How can data with different domains be integrated?
3. How do we assign probabilities to the various possibilities that emerge in integrating databases?
4. How can the merge results be represented in XML?
In order to simplify the problem, an assumption is made that a mapping between two databases already exists. This subject is part of other research and will not be covered here. Furthermore, we assume the databases to be XML databases. This structure is very flexible and it is therefore easier to add uncertainty [3]. Besides that, relational databases can always be translated to a XML database [2], so the solution applies to relational databases too.

This paper proposes methods which contribute to the solution to the problems as stated above. These methods will contain a description of how to merge data which is related via a one-to-many or many-to-many relations. Furthermore, it will describe some rules to handle with domain differences and several ways to store the various possibilities along with their probabilities in a XML format are discussed. Literature is reviewed in order to answer these questions.

1.2 Overview
In the next chapter relevant work are discussed. In chapter 3 the integration process will be discussed. An overview will be given of the whole process. In the chapters following, some specific problems are discussed, along with a way to represent uncertainty in XML.

The methods are implemented and validated with a subset of the EPrints and the Metis database. These databases both store data about published papers. Some of the records contain data about the same paper. Both databases are formatted with XML and have different schemata. Even the languages of the schemata are distinct. The records are synchronized on creation as of a certain year, but changes are not always synchronized between the libraries. This leads to inconsistencies and incompleteness. In section 7 this implementation and the outcomes are discussed.

2. RELEVANT WORK
Research on uncertainty in databases, or in particular in XML, has been done before. Magnani et al. did a survey on maintaining uncertainty in the process of data integration [5]. They also give a list of open problems. These problems include aggregation of matcher results and generation of mediated schemata. Although the last problem is more about the automatic generation of the schemata, it still remains an open problem.

Magnani also published a report about probabilistic data integration [6]. They describe an approach to integrate data, where the generated uncertainty is added in the resulting schemata. However this report mostly focusses on the generation of the mappings between the different databases, but little on the integration of the data itself.

Uncertainty in XML is subject of research conducted by Nierman and Jagadish [7] and Van Keulen et al [3]. Nierman and Jagadish propose an approach to represent uncertainty in XML. Mutual exclusion and incompleteness is taken into account, resulting in a proper structure useful in many situations. Van Keulen et al. propose a structure represented by a probabilistic XML tree, where probability, possibility and ordinary nodes alternate in a well-structured tree. Both structures will be discussed in more detail in chapter 6.

As shown in [6], preserving different possibilities with their probabilities can be used to provide richer answers to the users. On the other hand, keeping too many possibilities and mappings open increases the size of the database exponentially, as shown in [3]. This and other problems show there is still need for further research in this field.

After the integration of several databases, a database management system is needed which can handle uncertain data. We refer to MayBMS, a database management system which supports uncertainty, including uncertain queries. This system is described in more detail in [4].

3. INTEGRATION PROCESS
During the integration of the databases one of these databases is assumed to be the base database. This means that the records from the other databases are integrated into the base. The other databases are used to “enrich” or “improve” the base database. This simplifies some calculations and is nearly always the purpose of the integration. Besides this, it gives way to a structured way of choosing how to represent data, as this done by keeping as close to the structure of the base database as possible. Every database could be chosen as base, depending on the desired result. The base database is called A, the other is called B.

The mapping between fields, which we assume to exist, is represented as in the following definition

**Definition 1** A mapping between two schemata is represented by tuple containing two lists of fields of both schemata: \( <F_A, F_B> \), where \( F_A \) is a subset of the fields of database A and \( F_B \) is a subset of the fields of B.

In example 1, this would mean that the “name” field in A is mapped to the “name” field in B, whereas the “address” field in A is mapped to the “street” and “number” fields.

Because some fields could be mapped in an one-to-many or many-to-many mapping, additional steps are needed here in order to integrate these fields. A translation is needed where the fields are translated into a single set of fields. The mappings are reduced to simple one-to-one mappings and can then be integrated. This step is discussed in more detail later on. As concluded there, the best option is to translate the fields of database B to the fields of database A.

The integration process itself exists of five steps. The first step is applying the translation rules, so only one-to-one mappings are left. Step two and three are executed for every possible combination of objects from database A and B. In the second step, the probability of fields being equal are calculated for every corresponding field in A and B. The probability that the object of B refers to the same RWO as the object of database A is calculated in step three. In this step, the similarities of the previous step are used.

**Definition 2** The result of the integration is a set of tuples \( <a, b, p, \{<f_r, e_r>\}> \), where \( a \) and \( b \) are objects of database A and B, \( p \) is the probability these objects refer to the same RWO, \( f_r \) is a field in both A and B after translation and \( e_r \) is the similarity of these fields.

To save resources, combinations can be filtered out between steps. If the likelihood of two objects referring to the same RWO is very low, it is not useful to keep this combination. The same goes for important fields, like a person’s name.

---

**Figure 1. An overview of the integration process**

- **A**: Calculate similarities of fields
- **B**: Translate to schema of A
- **R**: Normalize probabilities
- **Export to XML**

- **Calculate similarity of reference to real world object**
- **Export to XML**
After all these steps, the results should be normalized to make the sum of all probabilities of the possible matches to a record from A equal to one. One-to-many or many-to-many relations should be handled differently. This will be discussed in chapter 5.

Once this is finished, the result should of course be exported to XML. An XML structure to save uncertainty is discussed in chapter 6.

4. DIFFICULT INTEGRATIONS

In data integration, two difficult integrations arise: the integration of one-to-many and many-to-many mappings and the integration of fields with different domains. Besides this, incomplete data could be an issue in the integration process. These problems are discussed in this chapter.

4.1 Mapping

In order to integrate data in one-to-many and many-to-many field mappings, both sets of fields need to be normalized into a single form. One of three possibilities could be chosen here: the first is translating one set of fields into the set of the other database. The second option is to create a set of fields “in between” the two sets. The third options is to reduce the sets of fields of both databases to a single field, which can then be integrated easily as a one-to-one relation.

For the case of this paper, the first options suits best. The second option introduces another database structure and would need to be translated back to the structure of the base database, as the integration is used to enrich this database. The third option would be useful and maybe it is the easiest solution. However, this options reduces the amount of fields and therefore makes searching or indexing more difficult. Next to this, some of the fields could be part of another mapping which would introduce redundancy.

That said, a translation is needed from the fields of database B to the fields of the base database A. Sometimes this translation splits a single field into multiple fields, in other cases fields are glued together. Finding such an translation automatically is as difficult as discovering the mappings between the fields and is topic of other research. In this paper this translation is therefore assumed to exist and it is set manually.

However, it could be the case that there are less fields in B than there are in A. Think for example of the same situation as in example 1, but database B does not longer contain the “number” field. In such situation you cannot translate the fields of B to the fields of A. The same reasoning applies if fields can be empty.

In this situation, the use of a wildcard could be used. If the house number is missing, it can simply be anything and any other possibility where the street is equal is considered the same. However if a field exists but is empty in a record, it could be this field is simply not known and the field being not empty in A could be an error. In this case the empty value is also an possibility with its own probabilities.

On the other hand, it could also be the case that B contains more information than A. This is however less problematic. A translation of fields from B to A can still be created. It depends on the situation and the implementer if this additional information is inserted in the resulting database.

4.2 Different domains

Fields from different databases could, although they store information about the same property, be stored as different types or in different domains. Information could contain errors simply because it doesn’t fit in the specified domain. Think of strings limited by a maximum length, or house numbers saved as numbers, so additions can’t be saved.

In order to handle different types, a conversion is needed. The conversion depends on the type of both fields. As a fallback, most fields can always be converted to a string, so it is unlikely such a conversion does not exists.

But what if, because of a wrong domain, a field contains erroneous values? If there exists a change a field is wrong, for example if the length of a string is near its maximum allowed length, the similarity calculation as described in the next chapter could be more flexible. The usage of a wildcard is possible too, to indicate the field could be equal to a field of a larger domain with more data. In case of numbers, one could think of an error marge. This is especially helpful in the case of double to float conversion and vice versa, as conversion errors are easily introduced here.

**Example 2** Due to a length restriction in database B of example 1, the length of a street has a maximum of 8. As a result, the record (“John”, “Mainstreet”, 3) is now stored as (“John”, “Mainstr.”, 3). As the length of “Mainstr.” is close to 8 and ends with a dot, the probability this field is shortened is high. When comparing it with “Mainstreet 1” from database A, it therefore gets a higher similarity probability.

4.3 Incompleteness

Records in a database could be incomplete. In XML, there are two different cases in which this can occur: elements can be empty or elements can be missing. In the first case it is known this object has this property, for example a name, but the actual value of it is not known. In the latter case, it is not known whether this object has this property at all.

In this research we assume both cases indicate that the value of the missing or empty element is not known, or that no valid value for this field exists. In data integration, somehow this incompleteness needs to be kept. If not, information would be lost. Think of a record where the actual value is not known, but by accident data is inserted in this field in one of the databases. This would lead to a probability of 1 if the empty fields would not participate in the distribution.

To handle this problem, empty fields have to be considered as possibilities too. However, a new problem arises here. What probabilities should be assigned to these empty fields? An probability based on similarity cannot be calculated, and other matches based on the values won’t work either. The probability assigned to this is discussed in chapter 5.1.4.

5. ASSIGNING PROBABILITIES

The easiest way of assigning probabilities is to assign equal probabilities to all possible combinations. However, this makes different combinations indistinguishable and leads to an exponential increasing of the database size, as shown in [3]. To avoid this problem, matchers are used in all state of the art integrations. Matchers give a combination of two objects from different databases a probability representing the likelihood they refer to the same RWO. This probability is most of the times a combination of the outcomes of different matches.

But how is this probability calculated? The obvious way is to have a calculation based on the similarities of the individual fields, as calculated in step two. Now this problem is reduced to finding the similarities of the fields or, to be more specific, of the fields of A and the translated fields of B.
5.1 Calculating similarities

The computation of the similarity results in a number $Eq$ in the range from 0 to 1, where 0 means not equal and 1 means equal. For each different type there are different calculations, since you can’t use mathematical calculations on strings.

5.1.1 Numbers

For numbers, simple subtraction or division is not sufficient, as this can easily result in a number not in the correct range between 0 and 1. To make sure the result is valid, we divide 1 by some exponent $c$ to the power of the difference between the numbers:

**Definition 3** The similarity of two numbers is calculated with

$$Eq = \frac{1}{e^{c\text{abs}(a-b)}}, \text{where } a \text{ and } b \text{ are the two numbers and } c \text{ is a constant.}$$

The constant $c$ defines the impact of the difference on the result and is dependent on the order of numbers you compare. The greater $c$ is, the faster the result decreases as the difference increases. A large $c$ is therefore used for very small differences.

5.1.2 Strings

In comparing strings, multiple matchers could be used.Matchers are divided in character based and token based implementations. Most used nowadays are edit distance and Jaccard similarity [1]. Edit distance is character based and calculates the minimum number of character changes (insertion, substitution or deletion) needed to transform one string in the other. Jaccard similarity looks at the number of common words in a string and divides this by the total number of words.

Of course there are other implementations available. Character based implementations are mostly used in short strings, whereas token based implementations are mostly used in long strings such as sentences.

5.1.3 One-to-many

In one-to-many or many-to-many relations, a field does not contain a single value, but a list of values. In the example used this paper, a publication has often more than one author. This could be represented as a list.

In assigning probabilities for one-to-many relations, every item in the list of B is compared with the items of A. If there is an equal item in A, both probabilities are added. If there is not an equal item in A, the value is added as a new possibility with its own probability. An example of this can be found in example 2.

5.1.4 Empty fields

A calculation based on the similarity or any other property of the fields cannot be done if one of the fields is empty. Therefore a static probability is needed to attach to empty fields. For the sake of simplicity, a probability of 1 is given to these fields. This can be justified by thinking of empty values as values that are unknown yet and thus can be anything, so anything is valid according to this possibility.

5.2 Calculating $p$

Now that the similarities for all fields are calculated, the probability both records refer to the same RWO can be computed.

**Definition 4** The probability $p$ two records refer to the same RWO can be calculated with

$$p = \frac{\sum w_i p_i}{\sum w_i}, \text{where } p_i \text{ is the similarity of a field and } w_i \text{ is a weight for this field}$$

In most cases the weight is simply one. However it could be that there are fields which are more important in this calculation. For example the titles of publications in a library are a better indication that two records refer to the same publication than the year of the paper. The weight is used to differentiate between fields.

5.3 Weighting similarities

An obvious action is to first filter out all non-matching combinations and all combinations with a very low $p$. If this is not done, you end up with a big database with a lot possibilities for each field, most of which will have a probability near zero. If database B has 1000 records, all fields in the resulting database will have 1001 possibilities.

Once the useless combinations have been removed the remaining combinations can be integrated into one database. To do so, all probabilities of the remaining possibilities of each field is multiplied with the computed $p$. This $p$ acts as a weight for the fields, because if it is less likely two records refer to the same RWO, the individual fields should also have a lower probability.

As one can conclude, all probabilities of database A are 1. The similarity of these fields compared to the base record is always 1, as they are simply the same. In computing the probability they refer to the same RWO, with probability 1.

The same calculation applies in the case of one-to-many or many-to-many relations. The difference in calculation between these and one-to-one is in the normalization step, as discussed in the next section.

**Example 2** A record in database A storing phone numbers of people has a record (“John”; [1234, 0612]). This record has been matched with (“John”, [1235, 0612]) in database B with $p=0.5$, while the phone number 1235 of B has an similarity of 0.75 and 0612 an similarity of 1.0. The resulting values are shown in table 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>1234 (p = 1.0*1.0 = 1.0)</th>
<th>1235 (p = 0.5*0.75 = 0.375)</th>
<th>0612 (p = 1.0<em>1.0 + 0.5</em>1.0 = 1.5)</th>
</tr>
</thead>
</table>

The resulting values do not sum up to 1, so they have to be normalized before they can be exported to XML. The next section discusses this.

5.4 Normalizing

The normalization of fields in a one-to-one relation is relatively simple. The probabilities are simply divided by the total sum. The name “John” in example 2 will therefore be normalized to 1.0.

However, in one-to-many or many-to-many relations, simple division by the total sum is not valid. The final probabilities do not need to sum up to 1. Moreover, this is not likely, as each possibility has its own independent chance on being linked to this particular object. Instead, each $p_i$ is divided by the sum of $p$, the probability records refer to the same RWO.

After normalization of the result of example 2, the final record is the one showed in table 2.
The results are according to the intuition, giving that the two databases are the only knowledge you have. If both databases tell you that John has the phone number 0612, he surely owns this number. However is some attribute is only present in one database, there is no guarantee the object does have this attribute. Note that in this result, it is possible for John to have either 1, 2 or 3 phone numbers, with changes 3/12, 7/12 and 1/12 respectively.

The combinations are now ready for exportation to XML. However, there are various options in how to save the records. This will be discussed now.

6. REPRESENTATION

Various representations for saving uncertainty in XML have been proposed. XML structures meant to contain uncertainty should have several properties. First, there should be a clear difference between certain data and uncertain data Secondly, it should be possible to include empty fields to support incompleteness. Lastly, it should be possible to indicate if a set of values is mutually exclusive, which means it should be possible to denote one-to-one, one-to-many and many-to-many relations.

Van Keulen and De Keijzer describe an probabilistic XML tree [3]. This tree consists of probability nodes, possibility nodes and ordinary nodes. The sum of probabilities from the children of a probability node is always 1. The tree is well structured, which means there is only one type of node on each layer. An example, taken from their paper, is shown in figure 2.

![Figure 2. An example of an probabilistic tree as described by van Keulen et al](image)

Although the well-structured property of this tree is easy to use, it gives much overhead. Every field is now nested in three elements instead of one. Some nodes, like the possibility node and the ordinary XML nodes could be taken together, as well as the probability node and the ordinary XML nodes.

There is even more overhead if a field is not mutual exclusive and thus can have a variable amount of values for a property. Every combination needs to be listed in this tier with the corresponding probability. If John for example has between 1 and 3 telephone numbers as in table 2, this would be represented

<table>
<thead>
<tr>
<th>Name</th>
<th>John (ps = 1.5/1.5 = 1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>1234 (ps = 1.0/1.5 = 0.67)</td>
</tr>
<tr>
<td></td>
<td>1235 (ps = 0.375/1.5 = .25)</td>
</tr>
<tr>
<td></td>
<td>0612 (ps = 1.5/1.5 = 1.0)</td>
</tr>
</tbody>
</table>

in this model with 7 different possibility nodes in which the three numbers are stored multiple times. So next to the overhead, this model also introduces a higher risk of inconsistency.

Nierman and Jagadish present a Probabilistic Tree Data Base (ProTDB) [7]. Elements can contain an attribute Prop, which contains the probability this element exists in the possible world, give that all parent nodes exist. The probability of the existence of B is therefore denoted as P(B|A).

If a field has a set of possibilities, it is denoted with a Dist element containing a set of Val children. The distribution can have a type attribute to indicate potential mutual exclusiveness. An example of this structure is showed in figure 3. A triangle represents a Dist node and an open circle represents a Val node. The example is a representation of the data in table 2.

![Figure 3. Graphical representation of the ProDTB XML structure](image)

As data could be incomplete, the sum of all probabilities within a distribution is somewhere between 0 and 1. The remaining 1-sum probability indicates the possibility of the field being empty.

As the last representation is the most complete, this representation is adapted in the implementation. Only one change is made: the probabilities in fields with one-to-many or many-to-many relations do not need to sum up to a number between 0 and 1, in order to represent the probabilities discussed in chapter 5.

7. EXPERIMENT

The integration process is implemented with two databases of different paper repositories on the University of Twente: the EPrints repository of the Faculty of EEMCS and the university-wide Metis repository. The EPrints repository contains mostly publications of the Faculty of EEMCS. The Metis repository contains most of these publications too, but some publications are only present in the EPrints repository. The data, however, is inconsistent as the information is only synchronized on creation as of a certain year. If after the insertion data is changed in one of the databases, differences exists between the data in both repositories.

This resulted in a lot of inconsistency between the repositories and in some cases incompleteness, as some fields empty on insertion were filled later on.

In the implementation, the EPrints repository is enriched with information of the Metis repository. This means the EPrints database is the base database (A). In the experiment, only publications in 2006 according to the base database are reviewed. This subset contains around 1200 publications.

Only the properties title, author, year and url. The author property has a many-to-many relation to publications. Each publication in the base database has an average of 3.2 authors. Sometimes the data is incomplete by means of a missing url or some missing authors. The problem of different domains is represented by the
year attribute, as the original EPrints database contains a date, whereas the Metis database contains a year represented by an integer. The date however is not always a full date, but most of the times only a year when the month and date are unknown.

In the application of weight factors to the fields, the title has a weight of 4, while the other fields have a weight of 1. This is because the title of a publication is a stronger indication whether two records referring to the same RWO than any of the other fields and after some tests, the weight of 4 seems to produce the best results.

After the integration process, two different databases are produced. The first contains uncertainty and therefore different possibilities, whereas the second result does not have uncertainty but simply the best possibility is chosen. In case of a one-to-many relation like the author field, the possibilities with probabilities above a certain threshold are included in the second database. The latter result represents the ordinary way of integrating two databases.

A typical result in the first resulting database is displayed in listing 1. As seen, two publications with identical titles have been found. However, the author and url fields differ. The author field is not mutual-exclusive and the probabilities do not add up to 1.

Several conclusions can be made from this result. It is most likely the paper was authored by 3 authors, as Visser Martijn and Visser M most likely refer to the same person. This is however not sure, so it is indeed correct both possibilities are saved.

Another conclusion which can be drawn is that this paper has two different urls. Both databases had a different url stored for this paper. In the current structure, the paper field is mutual exclusive. This might however need to be changed, as both urls seem legit.

Listing 1. A typical result of the integration process

A random subset of 20 publications is taken from both resulting databases. The results are manually checked on correctness, resulting in a true-negative percentage, indicating the percentage of data lost in the database without uncertainty, while it is actually valid data but it was simply not possible to export it. Another result is the false-positive percentage, indicating the amount of data included in the first result, while it is actually not a valid match. The outcomes are listed in table 3.

Table 3. False-positive and true-negative percentages in comparing the two resulting databases

<table>
<thead>
<tr>
<th>Field</th>
<th>False-positive</th>
<th>True-negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>5/25 = 20%</td>
<td>0/25 = 0%</td>
</tr>
<tr>
<td>author</td>
<td>5/98 ≈ 5.1%</td>
<td>13/98 ≈ 13.3%</td>
</tr>
<tr>
<td>year</td>
<td>1/21 ≈ 4.8%</td>
<td>0/21 = 0%</td>
</tr>
<tr>
<td>url</td>
<td>6/46 ≈ 13.0%</td>
<td>20/46 ≈ 43.5%</td>
</tr>
</tbody>
</table>

As seen, the matching process is not perfect. It also matches some records, which are actually not referring to the same publication. Especially the title field contains a lot of mismatches, which can be explained by the fact that there exists much different publications with the same set of authors. Improved matchers or better weighting values could lower the false-positive rates and is therefore subject to further research.

On the other hand, by using the ordinary integration result without uncertainty in it, a lot of data is lost. In particular in the url field a lot of data is lost. Around 43.5% of all valid urls in this subset is lost in this integration result. In some cases, the result did not contain any url at all, which happens if the possibility of having no url has the highest probability. In the result this will be interpreted as unknown, while in reality there was an url known.

Next to this, in the author field a lot is lost too, although a remark should be made that much of the lost authors are still present in the result in the form of differently spelled names.

Overall, it is more preferable to keep uncertainty in the result of the integration process. Although the result contains some mismatches, the loss of data using the method without uncertainty is worse than these false-positives.

8. CONCLUSIONS

One-to-many and many-to-many mappings can be integrated with the use of translations, which translate the fields to the schemata of the base database. If this does not apply, wildcards could be used. The same rule applies to different domains in mappings, although sometimes it is not possible to translate the fields. In this case error margins or wildcards are helpful, but the domains need to be known in order to define these.

Probabilities can be assigned in different ways. The most appropriate way is to calculate the similarities between fields. The resulting similarities are weighted with the probability both records point to the same real world object and normalized afterwards.

In the representation of the uncertainty in XML, the structure proposed by Nierman and Jagadish, called ProTDB is used, but slightly changed. This structure supports mutual exclusive values and incompleteness, and is therefore a proper format.
The resulting database still contains false-positives, but this can be improved by using better matchers. On the other hand, using the ordinary way to integrate databases without uncertainty, a lot of data is lost, especially if there exists much differences between the two databases which are integrated.

9. REFERENCES


