A METHOD FOR EARLY CORRESPONDENCE DISCOVERY USING INSTANCE DATA

Indrakshi Ray and C. J. Michael Geisterfer
Department of Computer Science, Colorado State University, 601 S. Howes Street, Fort Collins, USA

Keywords: Information Integration.

Abstract: Most of the research in database integration have focussed on matching schema-level information to determine the correspondences between data concepts in the component databases. Such research relies on the availability of schema experts, schema documentation, and well-designed schemas – items that are often not available. We propose a method of initial instance-based correspondence discovery that greatly reduces the manual effort involved in the current integration processes. The gains are accomplished because the ensuing method uses only instance data (a body of database knowledge that is always available) to make its initial discoveries.

1 INTRODUCTION

The need to integrate information from different databases arise in many scenarios. For instance, the latest trend for organizations is to purchase applications from a outside software company targeted at specific business functions and then integrate these multiple business applications into an organization-wide operations support system. The result is the partitioning of the data of the organization along with the partitioning of the business functionality. These applications will work with their own data and data models, optimized for the specific business functionality. Although these different applications use their own data models, they will most likely work on data representing the same real-world concepts and therefore will contain some intersection of similar data. Since these applications were developed by independent vendors, they will not share the same schema. Ultimately, the schemas and data of these various component databases must be integrated in some manner before the data can be accessed and manipulated across the greater domain at the organization-wide level.

In the database integration research (Aslan and Mcleod, 1999; Chua et al., 2003; Lawrence and Barker, 2001; Lenzerini, 2002; Parent and Spaccapietra, 1998; Rahm and Bernstein, 2001; Schmitt and Trker, 1998; Yan et al., 2002; Zhang, 1994), much effort has gone into developing automated solutions to integrating the schema (and then, afterwards, the data itself). In most of the research literature, the solutions and approaches have concentrated on matching schema-level information to determine the correspondences between data concepts in the component databases. If instance-level information is even utilized, it is used only to augment the correspondences found using the schema-level information, to catch what the schema-level matching missed.

In most of the relevant research, the schema transformation step occurs *before* correspondence investigation. It is argued that the schemas must be transformed into a canonical data model before the schemas can be analyzed for correspondences (Aslan and Mcleod, 1999; Parent and Spaccapietra, 1998; Rahm and Bernstein, 2001). Schema transformation itself can be an involved process, and although there are a few tools available, this too is a manual process relying again on the expert knowledge of the schemas involved.

This paper challenges both these mainstream positions by considering an instance-based initial inter-database correspondence discovery method that greatly reduces the manual effort involved over the entire integration process. Because this method is applied *before* schema transformation, it saves effort by

Ray I. and J. Michael Geisterfer C. (2007).
A METHOD FOR EARLY CORRESPONDENCE DISCOVERY USING INSTANCE DATA.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 258-263
DOI: 10.5220/0002350102580263

providing for an more efficient transformation process at a later step. Because it initially uses instance-level information, lack of expert schema-knowledge and poor schema designs do not hinder its progress.

We propose that schema transformation occur *after* correspondence discovery. If one determines (discovers) the relationships between the schemas of only those portions of the databases that are to be integrated, then one has determined the portions of the schemas that must be integrated. Once those portions of the schemas have been identified and related, they can be transformed to meet the integration model desired. There is now less schema to transform, and the actual transformation process is simpler because there is know exacting knowledge of the schemas involved. The main contribution of this paper is to propose a method of initial instance-based correspondence discovery that greatly reduces the manual effort involved in the current integration processes. The gains are accomplished because the ensuing method uses only instance data (a body of database knowledge that is *always* available) to make its initial discoveries.

The rest of this paper is organized as follows: section 2 explains the method; and section 3 concludes our paper and provides some pointer to future work.

2 THE METHOD

The proposed method focuses on correspondence investigation – determining correspondences between two or more databases. We require that the target databases be *intersected* as we will be searching the intersection area for (nearly) exact matches of information. Two databases are said to be intersected if the real world concepts they represent have some common elements or are otherwise interrelated. An example of intersected databases would be two separate applications, one an order management system and the other a billing system, that each need to integrate the concept of a customer. In such cases two elements from two databases *correspond to each other* (Parent and Spaccapietra, 1998) because they describe the same real world concept or fact.

Our approach basically involves two activities: defining potential correspondences and accepting those correspondences validated by the method. But first, we need to discuss the matching of instance data.

2.1 Instance Matching

Our goal is to find out if two attributes are related to each other. For this, we need to compare, or *match*, these sets of instance values using only the type of data and *not* its meaning (semantics). We execute our initial matching on raw instance data *specifically* ignoring any schema information.

We define a function called the *RawMatch* that checks whether the the attributes of tables belonging to heterogeneous databases are equal or not. This function compares the data in the tables looking for exact or nearly exact matches. Nearly exact matches are defined as similar data that meets a specified confidence factor. *Confidence factor*, expressed as a value in the range [0..1], defines the amount of difference that can be tolerated when comparing two items *A* and *B*, where *A* and *B* can be simple instance data, attributes, or sets of attributes. The confidence factor is selected by the integration specialist based upon his/her idea as to the similarities between the data in the databases.

Algorithm 1 Determine if two instance data match.
Input: (i) d_i, d_j – instance data and (ii) f – confidence factor.
Output: boolean – indicates whether or not a match has been found
Proc RawMatch(d_i, d_j, f)
if compare(d_i, d_j) $> f$ **return** true
else return false

The procedure *compare* compares between two data instances and calculates a matching factor. *Matching factor*, expressed as a value in the range [0..1], indicates the amount by which two elements *A* and *B* match where *A* and *B* can be simple instance data, attributes or sets of attributes. For exact matches between two data instances d_i and d_j, *m* equals 1. For no match, *m* equals 0. The exact procedure *compare* depends on the type of data instances. We show what this procedure does for string type and numeric data.

Comparing String Type Instance Data For character string based data, comparisons are made string-to-string, case insensitive, with and without spaces removed. If a difference is detected, *RawMatch* will attempt to re-establish alignment (shift) in the strings (e.g., a simple case where the first string uses a space to separate characters and the second string does not). For each comparison, a matching factor, *m*, is calculated. For exact matches, *m* = 1.000. For nearly exact string matches, *m* is calculated as a percentage of the string that was similar. For example, when comparing the two strings:

$s1$ = "`1024 West Valley View Rd.`" and
$s2$ = "`1024 West Valleyview Rd`",

RawMatch will find an initial difference at position 17, where $s1$ = `' '` and $s2$ = `'v'`. It will the try to

re-establish alignment, finding it again at the next position (so now it has reset itself and is comparing $s1$ = 'V' and $s1$ = 'v'. It finds the next difference when $s1$ = '.' (because $s2$ doesn't have anymore characters). Now the strings are exhausted.

So, with strings of length 25 (maximum of the two), and with 2 mismatches, the *compare* function computes the matching factor *m* in the following way: it calculates the percentage of matching characters in the string, that is, m = (23/25) = .920. If this value is above the pre-determined confidence factor f, then RawMatch counts this as a match between 2 instances of attributes for tables from different databases.

Comparing Numeric Instance Data Numeric values are either exact matches or can be a near match. The matching factor can be calculated by using statistical functions.

2.2 Correspondence Investigation

We try to find the relationships between the attributes belonging to tables in different databases. An example will help to illustrate this. Let element e_1 refer to an instance value for the attribute `Name` in a table in database A, and let elements e_2 and e_3 refer to instance values for the attributes `FName` and `LName` respectively, in a table in database B. If the values of e_1, e_2, and e_3 are "John Henry", "John", and "Henry", respectively, then we would say that e_1 *corresponds in an equal manner* to the concatenation of e_2 and e_3.

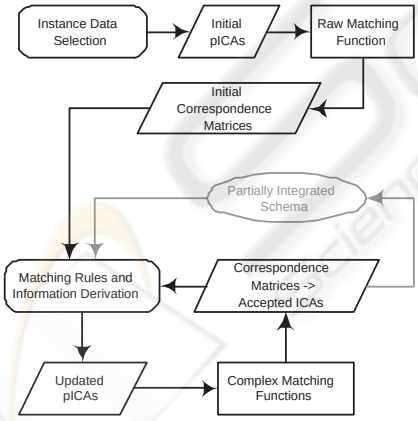

Figure 1: Instance-based Correspondence Discovery Method.

If we get enough of such instance to instance matches, then we can become more confident that the tables containing those fields correspond to each other. The "enough" is specified by the value of the confidence factor F, supplied by an integration specialist. The task is to determine how many times values from attributes in one database matches values from those in another database. Sometimes transformation functions need to be applied to the attributes before they can be compared. The function $transform$ changes the data to the correct form before they are input to the *RawMatch* algorithm for comparison. Using the function *RawMatch* we count the number of instance matches over the total number of instance (number of rows in the tables). This will generate a raw percentage: if 87 out of 100 instance pairs matched then the matching factor M for the attributes, is given by, M = .870 (87/100). If the matching factor M is greater than F, then a correspondence does exist between the sets of attributes.

Table 3: Table Type in DB2.

id	T1	T2	TxCd
1	1	0	1
2	1	1	1
3	2	0	3
4	2	1	4
5	3	0	8
6	3	1	12
7	4	0	2
8	4	1	2

Algorithm 2 Correspondence Discovery Algorithm.
Input: (i) $\{A\}$ – set of instance data corresponding to attributes of one database, (ii) $\{B\}$ – set of instance data corresponding to attributes of second database, and (iii) F – Confidence factor
Output: boolean – indicating whether a correspondence exists between the two sets of attributes

Proc CorrespondenceInvestigate(A, B, F)
matchInstance = 0
totalInstance = 0
for each a_i in A **do**
  totalInstance = totalInstance + 1
  for each b_j in B **do**
    if $RawMatch(transform(a_i), transform(b_j), f)$
      matchInstance = matchInstance + 1
$M = matchInstance/totalInstance$
if $M > F$ **return** true
else return false

2.3 Method Synopsis

Our method is described in Figure 1.

2.3.1 Instance Data Selection

The initial step is the selection of the initial instance data which needs to be matched to discover correspondences. When looking at the areas of interest in

Table 1: Table CustType in DB1.

id	Type	Cname	Fname	Mname	Xname	NamePrfx	PriAddr
0	Res	Smith	David	A		Mr	1
1	Res	Johnson	Elaine	A		Ms	3
2	Bus	Redding	Alan	D	DDS	Dr	8
3	Edu	Dept CS			CSU		15

Table 2: Table Addr in DB1.

id	Cid	Num	Prefix	Street	Type	AptT	AptN	City	State	Post
1	0	123	West	57th	Ave			Ft. Col.	CO	80524
2	0	4167	North	Reading	Way	Apt	B	Ft. Col.	CO	80521
3	14	8008		Hampden	Rd	Suite	101	Ft. Col.	CO	80523
4	1	1001	East	Main	St	Suite	220	Laporte	CO	80544

the component databases to be integrated, one must identify which portions of these databases have *potential* intersections of data. If, for example, the choice has been made to integrate customer and product information, one only needs to concern themselves with those portions of the databases that contain information (instance data) about those topics (concepts). A high-level perusal of the schemas and data in these databases should yield a reliable idea of which tables contain data concerning customer and product. For instance, we see that the Tables 1 and 2 in Database 1 have an intersection with Tables 3 and 4 in Database 2. This step should identify *any* tables that contain relevant information. This step does not have to be an exacting, exhaustive search in a particular database for all customer or product instances. This is not an initial step of inter-database relationship discovery. All that is being accomplished here is the identification of specific portions of a database that contains tables that are of interest to the concept areas we are trying to integrate.

Once these areas have been identified, the next step is to find correspondences at the table (relation) level because it will be these relations that will provide the initial data for comparison. We will not discuss this as it has been addressed by other researchers.

2.3.2 Inter-Database Correspondence Assertions

Using the information obtained during information discovery, we develop inter-database correspondence assertions (ICAs) over which our matching algorithms can operate. Initially, we do not have any previously matched relationships, so we can only specify tests. These tests are referred to as *potential* ICAs or *p-ICAs*. A typical p-ICA consists of two parts that are separated by the symbol $\Diamond$. This basically indicates that all the data in the tables (relations) to the left of the $\Diamond$ should be compared to all the data in the tables to the right of the $\Diamond$. An example p-ICA that asks to match data in the tables CustType and Addr in DB1 to the tables Type and Cust in DB2 is given below.
$\{DB1 :: CustType, DB1 :: Addr\} \Diamond \{DB2 :: Type, DB2 :: Cust\}$

The data is then analyzed by our matching algorithms. The output is a *correspondence matrix* that contains the confidence factors comparing the attributes of two tables. Using the data from Tables 1, 2, 4 and 3, one of the correspondence matrices produced by our matching algorithms is given in Table 2.3. (The factor f for string matches was set at 0.50).

2.3.3 Matching Rules

The next step is to analyze the correspondence matrix to identify relationships. At the completion of the analysis, more ICAs will have been discovered. These new ICAs will be the input for the next round of matching.

Analysis The integration specialist must analyze the correspondence matrices and decide which relationships are genuine, which are false, and more importantly, which potential relationships are worth pursuing. This will help in determining the new ICAs. Accepted correspondences become new ICAs; rejected matches are explicitly stated as inequality ICAs. This will prevent the match functions from "revealing" these matches again in subsequent processing. By analyzing Table 2.3 above, we can gather that there are strong relationships between the following attributes:

```
DB1::CustType::ID          ≅  DB2::Cust::ID
DB1::CustType::Cname       ≅  DB2::Cust::Lname
DB1::CustType::Fname       ≅  DB2::Cust::Fname
DB1::CustType::Mname       ≅  DB2::Cust::M
DB1::CustType::Xname       ≅  DB2::Cust::Other
DB1::CustType::NamePrfx    ≅  DB2::Cust::Sal
```

Table 4: Table Cust in DB2.

id	ty	Sal	Fname	M	Lname	Oth	Num	StrNam	Apt	City	St	Zip
1	1	Mr	David		Smith		123	57th		Ft. Col.	CO	80524
2	1	Ms	Elaine		Johnson		1001	Main	220	Laporte	CO	80544
3	7	Dr	Alan	D	Redding	DDS	321	Medical	200	Ft. Col.	CO	80522
4	8		Howard	X	Hughes		1	Hughes		Bellevue	CO	80558

Table 5: Correspondence Matrix between DB1::CustType and DB2::Cust.

	DB2:Cust								
DB1::CustType	ID	Type	Sal	Fname	M	Lname	Other	Num	...
ID	0.750	0.425	0.000	0.000	0.035	0.000	0.000	0.025	...
Type	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...
Cname	0.000	0.000	0.000	0.000	0.000	0.750	0.000	0.000	...
Fname	0.000	0.000	0.000	0.750	0.000	0.000	0.000	0.000	...
Mname	0.000	0.000	0.250	0.000	0.750	0.000	0.000	0.000	...
Xname	0.000	0.000	0.000	0.000	0.000	0.000	0.750	0.000	...
NamePrfx	0.000	0.000	0.750	0.000	0.250	0.000	0.000	0.000	...
PriAddr	0.315	0.425	0.000	0.000	0.000	0.000	0.000	0.175	...

while there exist some possible relationships between the following:

```
DB1::CustType::ID          ≅  DB2::Cust::Type
DB1::CustType::ID          ≅  DB2::Cust::M
DB1::CustType::ID          ≅  DB2::Cust::Num
DB1::CustType::Mname       ≅  DB2::Cust::Sal
DB1::CustType::NamePrfx    ≅  DB2::Cust::M
DB1::CustType::PriAddr     ≅  DB2::Cust::ID
DB1::CustType::PriAddr     ≅  DB2::Cust::Type
DB1::CustType::PriAddr     ≅  DB2::Cust::Num
```

Creation of More Match "Rules" In our analysis, we will determine which discovered matches we will accept. For those for which a strong correspondence was found, we need to add these as ICAs. This will yield statements like:
$\{DB1 :: CustType :: CName\} = \{DB2 :: Cust :: Lname\}$
These statements must be added to the input file for the match function. If there was a match that we would like to not be used by the match algorithm (a match that we do not accept), for example to apparent match between the ID fields, then we need to place create an ICA indicating an inequality:
$\{$`DB1::CustType::ID`$\} \neq \{$`DB2::Cust::ID`$\}$
This inequality specification will indicate to the match functions that this relationship need not be processed. This will save some time in processing. We must analyze the weaker matches too, to see if there is a possibility that there are indeed good candidates for potential matches here. For the example given, there are no such situations.

More complex comparison "rules" can now also be added. For example, in the correspondence matrix comparing DB1::CustType with DB2::Type, we notice that there is no relationship established between the attributes (other than ID which we have already discounted). We notice that CustType uses a character field to indicate customer type while Type uses integer values. To overcome this, an ICA can be developed to give the match processing hints:
$\{DB1 :: CustType\} \Diamond \{\{DB2 :: Type\} AND$
$\{(\text{"}Res\text{"} = 1, \text{"}Bus\text{"} = 4, \text{"}Gov\text{"} = 2, \text{"}Edu\text{"} = 3)\}\}$
This specification lets the match processing know that there may be a relationship between these values. Another situation that must be matched would be looking for relationships between tables based on a single key field that pulls all the information together from multiple tables. This might be the case for the following specification:
$\{DB2 :: Cust\} \Diamond \{\{DB1 :: CustType, DB1 :: Addr\}$
$WHERE$
$\{DB1 :: CustType :: ID\text{=}DB1 :: Addr :: CID\}\}$
One could also search for a relationship amongst tables by keying on the particular value of an instance of an attribute:
$\{\{DB2 :: Cust\} WHERE$
$\{DB2 :: Cust = \text{"}Smith\text{"}\}\} \Diamond \{\{DB1 :: CustType, DB1 :: Addr\}$
$WHERE$
$\{DB1 :: CustType :: ID = DB1 :: Addr :: CID\}\}$
These ICAs can be come quite complex, allowing for a powerful correspondence discovery method.

2.3.4 More on Matching Algorithms

As the match processing progresses and the ICAs become more complex, we no longer find matches based solely on the instance data. Early corre-

spondence discoveries lead to true knowledge about the relationships between the actual schemas from both databases. This relationship knowledge is extremely reliable because it has been extracted from the database themselves, through the instance data. As our knowledge of these relationships increases, we can specify more exacting and detailed ICAs, resulting in even more knowledge. As the ICAs become more complex, and begin to rely increasingly on schema-level information, the match functions must grow in complexity accordingly. There are several good match algorithms (Aslan and Mcleod, 1999; Chua et al., 2003; Castano et al., 2001; Gal et al., 2003; Lawrence and Barker, 2001; Li and Clifton, 2000; Schmitt and Trker, 1998) that could be used at this stage in the process.

2.4 The Result

The correspondence investigation method proposed provides an initial discovery of inter-database relationships. During the implementation of the method, both information about the individual schemas of the component databases as well as initial inter-schema relationship are discovered. Accepted ICAs provide direct inter-schema relationship information. With only a portion of the component databases schemas needing transformation, the information in the accepted ICAs can be used in building this partial integrated schema. The existence of a partially integrated schema aids the integration specialist in detecting new assertions (pICAs) as well as errors with the existing ICAs. The result is enough information to make the approaches based on schema integration more viable and efficient.

3 CONCLUSIONS

This paper has proposed a method for correspondence investigation that does not suffer if there is a lack of expert knowledge of the schemas involved nor does it assume that those schemas are well designed. Implementation of this method will greatly reduce the manual effort involved in the integration process, which currently heavily dependent upon such efforts.

In future, we plan to integrate existing match functions developed in the artificial intelligence field with our work. Sophisticated and efficient match functions used in comparing instance data may give us more correspondence than would be otherwise possible. In future we would also like to develop a language to formally specify the inter-correspondence assertions.

ACKNOWLEDGEMENTS

The work was supported in part by a grant from AFOSR under Award No. FA9550-04-1-0102.

REFERENCES

Aslan, G. and Mcleod, D. (1999). Semantic heterogeneity resolution in federated databases by metadata implantation and stepwise evolution. *The VLDB Journal*, 8:120–132.

Castano, S., Antonellis, V. D., and di Vemercati, S. D. C. (2001). Global Viewing of Heterogeneous Data Sources. *IEEE Transactions on Data Knowledge and Engineering*, 13(2):277–297.

Chua, C. E. H., Chiang, R. H. L., and Lim, E. (2003). Instance-based attribute identification in database integration. *The VLDB Journal*, 12:228–243.

Gal, A., Trombetta, A., Anaby-Tavor, A., and Montesi, D. (2003). A model for schema integration in heterogeneous databases. In *Proceedings of the 7th International Database Engineering and Applications Symposium IDEAS '03)*, pages 2–11.

Lawrence, R. and Barker, K. (2001). Integrating relational database schemas using a standardized dictionary. In *Proceedings of the 2001 ACM Symposium of Applied Computing*, pages 225–230.

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–246.

Li, W. and Clifton, C. (2000). SemInt: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Network. *IEEE Transactions on Data Knowledge and Engineering*, 33(1):49–84.

Parent, C. and Spaccapietra, S. (1998). Issues and Approaches of Database Integration. *CACM*, 41(5):166–178.

Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350.

Schmitt, I. and Trker, C. (1998). An incremental approach to schema integration by refining extensional relationships. In *Proceedings of the 7th International Conference on Information and Knowledge Management*, pages 322–330.

Yan, G., Ng, W. K., and Lim, E. (2002). Product Schema Integration for Electronic Commerce A Synonym Approach. *IEEE Transactions of Knowledge and Data Engineering*, 14(3):583–598.

Zhang, J. (1994). A formal specification model and its application on multidatabase systems. In *Proceedings of the 1994 Conference of the Centre for Advanced Studies in Collaborative Research*, pages 76–89.