Coordination of Data in Heterogenous Domains

Michael Lawrence #1, Rachel Pottinger #2, Sheryl Staub-French *3

#Department of Computer Science, University of British Columbia
201-2366 Main Mall Vancouver, BC, Canada V6T 1Z4
[1]mklawren@cs.ubc.ca [2]rap@cs.ubc.ca

*Department of Civil Engineering, University of British Columbia
2002-6250 Applied Science Lane Vancouver, BC, Canada V6T 1Z4
[3]ssf@civil.ubc.ca

***Abstract*— Existing semantic integration approaches to coordinating data do not meet the needs of real world scenarios which contain fine-grained relationships between data sources. In this paper, we describe extensions to the popular GLAV mapping formalism to express such relationships. We outline methods for solving the data coordination problem using these mappings, and discuss future research problems for data coordination to be realized in heterogeneous domain scenarios that occur in practice.**

I. Introduction

In many applications, multiple heterogeneous data sources need to coordinate data so that changes made to one data source are reflected in another related data source. This allows the manager of a data source to ensure it is up to date and consistent with the latest data provided by related sources upon which it depends. Current information integration methods do not address some needs of real world data coordination problems, such as when the sources of data describe different, but related, types of objects. Such situations often have a small intersection between schemas and lack of a common domain. The implication is that there are very fine grained and complex relationships between values, rather than the commonly considered coarse grained relationships between tables. For example, in the architecture, engineering and construction (AEC) industry, these properties are exhibited by the relationships between a building's design and its cost estimate. Such relationships often require aggregation, arithmetic, conditional expressions etc. This is further illustrated in Section II. Current methods relying on a significant schema overlap and tuple level correspondences fail to meet the needs of these situations.

This paper describes our ongoing research into data coordination between sources with heterogeneous domains. We formally state the data coordination problem, and show how value correspondences between sources can be expressed by augmenting existing mapping formalisms with keys and mapping tables. We outline the proposed method for solving data coordination, state the remaining research challenges, and describe a number of extensions which address an even greater variety of real world problems.

The rest of this paper is organized as follows. Section II describes a motivating example, based on our work with practitioners in AEC. Section III formulates the data coordination problem, gives the details of our contributions and outlines the proposed algorithm for solving it. We discuss remaining research problems and extensions in Section IV. Related work is in Section V, followed by our conclusions in Section VI.

```
Wall(id, type, length, height, area)
Layer(id, material, thickness)
HasLayer(id, wid, lid)
```

Fig. 1. A subset of the schema for building design data.

```
CostItem(code, description, unit, rate)
Estimate(code, description, qty, unit, rate)
```

Fig. 2. A relational schema for cost items and a specific building's estimate.

II. A Motivating Example: The ARTIFACT Project

Our study of data coordination in heterogeneous domains arises as part of a collaborative project. The Advanced Research, Techniques, and Informatics for Future Advantages in Construction Technology (ARTIFACT) project is a collaboration between industry, computer science researchers, and civil engineering researchers. The project addresses information management issues which arise in real world Architecture, Engineering and Construction (AEC) projects.

One particular challenge arises in the context of cost estimating a large building project; maintaining a cost estimate as changes are made to the design is time consuming. Not only is it difficult to determine how a design has changed, determining the effects on the estimate is error prone and subjective. The design is frequently revised during the life cycle of a project; estimating errors are a major cause of revenue loss for general contractors. Consider the following example:

Example 1: In large AEC projects, 3D building designs are created and maintained by the principal architects. These designs consist of building components such as walls, columns and doors, and may be exported to a relational database. The designs are examined by a general contractor (GC) who creates and maintains a cost estimate. A subset of the (simplified) design data schema is shown in Figure 1, while the GC's cost data is structured as shown in Figure 2.

The GC would like to coordinate the cost data with the architect's design data so that changes are propogated automatically. For example the GC would like to express how the quantity, and cost of particular estimate items depends on aggregate functions of the design. □

III. Mappings for Data Coordination

In this section we formulate the data coordination problem using global and local as view (GLAV) mappings [1], which have previously been applied to data integration. We demonstrate with an example how GLAV mappings, although appropriate for expressing constraints, are ambiguous in the data coordination cases we address. We describe extending these mappings by using join keys and mapping tables, allowing a unique solution to be found for data coordination where complex value mappings are necessary. We outline our process for using the extended mappings to perform data coordination.

For a database instance $\mathbf{I}$ and query q, let $q(\mathbf{I})$ denote the relation resulting from executing q on $\mathbf{I}$, and $schema(q)$ denote the schema of the relations in q's domain. A GLAV mapping is of the form $q_I = q_J$ where q_I and q_J are queries, and represents a *constraint* which is satisfied with respect to database instances $\mathbf{I}$ and $\mathbf{J}$ if $q_I(\mathbf{I}) = q_J(\mathbf{J})$. Data coordination involves finding updates to $\mathbf{J}$ so that a set of mapping constraints are satisfied. Let $\Delta(\mathbf{J})$ denote the result of performing a set of updates Δ on a database instance $\mathbf{J}$. Formally, the data coordination problem is as follows.

Definition 1: (Data Coordination Problem) Given a set of mapping constraints $\Sigma = \{q_I^1 = q_J^1, \ldots q_I^n = q_J^n\}$, and database instances $\mathbf{I}$ and $\mathbf{J}$, find a set of update statements Δ_J such that $q_I^i(\mathbf{I}) = q_J^i(\Delta_J(\mathbf{J}))$ for all i. □

A. Solution Overview

In this paper we propose to execute each q_I^i and q_J^i, compare the results to obtain a set of updates Δ_q so that $q_I(\mathbf{I}) = \Delta_q(q_J(\mathbf{J}))$, and then translate Δ_q into Δ_J. Our final solution is the union of all such updates for each mapping constraint. This section describes some details of the solution. Consider the following example, which is based on design and estimate data from a real world construction project provided to the ARTIFACT team members by Stuart Olson Construction.

Example 2: A building design contains walls of various materials: w1 and w2 are interior partition walls consisting of two layers of gypsum wall board mounted on a layer of metal studs; w3 has a single layer of gypsum wall board mounted to a layer of concrete via metal furring; and w4 is only concrete. The data corresponding to the walls of this design is shown in Table I. An estimator has created a cost estimate for this design as shown in Table II; each item corresponds to a different type of wall. The category code 3310 is used for the material and labour items pertaining to structural concrete, while the category code 9250 is used for partition walls. The quantity of item "Pump and Place Wall Concrete" depends on the volume of w3 and w4; the quantity of "Metal Stud Partition Wall" depends on the area of w1 and w2, while the quantity of "Furred Partition Wall" depends on the area of w3. This estimator would like to express the quantities of partition walls in the estimate using a mapping constraint, and keep these quantities updated as walls are added, deleted or modified. □

We can express the cost estimator's mapping as $q_I = q_J$, where q_I and q_J are as in Figure 3. Note the complexity of q_I:

Wall

id	type	length	height	area
w1	1234	20	8	136
w2	1234	10	8	80
w3	1240	15	8	120
w4	1122	25	10	202

Layer

id	material	thickness
l1	"Gypsum Wall Board"	16
l2	"Metal Studs"	150
l3	"Concrete"	300
l4	"Metal Furring"	22

HasLayer

id	wid	lid
1	w1	l1
2	w1	l2
3	w1	l1
4	w2	l1
5	w2	l2
6	w2	l1
7	w3	l3
8	w3	l4
9	w3	l1
10	w4	l3

TABLE I

Example design data (**I**) consisting of 4 walls.

Estimate

code	description	qty	unit	rate
3310	"Pump and Place Wall Concrete"	20	cuyd	19.00
9250	"Metal Stud Partition Wall"	216	sqft	9.00
9250	"Furred Partition Wall"	120	sqft	9.00

TABLE II

Example cost estimate (**J**) for the design data in Table I.

a nested query must first select all partition walls (i.e. having at least 1 layer of "Gypsum Wall Board"), which are then aggregated by type. To our knowledge, most previous work does not allow mappings of this type. We can easily confirm that this mapping constraint is satisfied with respect to $\mathbf{I}$ and $\mathbf{J}$ by evaluating $q_I(\mathbf{I})$ and $q_J(\mathbf{J})$, which both yield a set of two tuples: $\{\langle qty : 216\rangle, \langle qty : 120\rangle\}$. However, if we evaluate the queries and get differing results (for example, $q(\mathbf{I}) = \{\langle qty : 230\rangle, \langle qty : 120\rangle\}$), there are many possible Δ_J for which $q_I(\mathbf{I}) = q_J(\Delta_J(\mathbf{J}))$. In this case, we could update the quantity of item "Metal Stud Partition Wall" from 216 to 230, or change the quantity of "Furred Partition Wall" to 230, and the quantity of "Metal Stud Partition Wall" to 120. In this example, the first update option reflects the estimator's intentions while the second does not, and the mapping constraint $q_I = q_J$ is not strong enough to choose the correct solution.

As shown in this example, there is a mismatch between the strict requirements of data coordination and the coarse relationships of GLAV mappings. A mapping constraint $q_I = q_J$ relates *sets* of tuples, whereas data coordination requires knowing which *tuples* are related. This is because data coordination requires updating individual tuples of existing and independently maintained data sources, whereas data exchange generates a data instance. In addition, when $\mathbf{I}$ and $\mathbf{J}$ are domain heterogeneous, the attributes selected by q_I and q_J typically describe but do not identify tuples (e.g. the attribute qty in Figure 3), and hence it is generally impossible to identify which tuples are related based on these queries alone.

B. Augmented Mappings

We propose to deal with update ambiguity by using *augmented* mappings. For a mapping constraint $q_I = q_J$, an augmented mapping constraint consists of 1) an expression of the form $q_I' = q_J'$, where $schema(q_I') = schema(q_I) + [k_I]$, and $schema(q_J') = schema(q_J') + [k_I]$ (k_i and k_j are called

```
SELECT SUM(Wall.area) AS qty
FROM (
  SELECT Wall.area, Wall.type,
         DISTINCT Wall.id
  FROM Wall
  JOIN HasLayer, Layer
  ON Wall.id = HasLayer.wid,
     Layer.id = HasLayer.lid
  WHERE material
    = 'Gypsum Wall Board')
GROUP BY type;
```

(a)

```
SELECT qty
FROM Estimate
WHERE code = 9250;
```

(b)

Fig. 3. The queries q_I and q_J of the example mapping constraint $q_I = q_J$. (a) returns the sum of partition wall areas, group by type. (b) returns the quantity of partition wall items.

mapping keys), and 2) a mapping table K [2], [3], which is a relation over the schema $[k_I, k_J]$. Currently, we require the mapping designer to choose keys and create augmented mappings. Part of our ongoing research involves suggesting candidate keys. The mapping key k_I should be such that for any feasible $\mathbf{I}$, each tuple in $q'_I(\mathbf{I})$ has a distinct value for k_I (similarly for k_J, $\mathbf{J}$ and q'_J.) The keys of an augmented mapping need not necessarily be keys of the databases.

Following Example 2, we can use the augmented mapping constraint $q'_I = q'_J$, where q'_I is identical to Figure III-A (a) except containing the term "Wall.type" in the SELECT clause, similarly for q'_J in Figure III-A (b) with the term "Estimate.description". A suitable mapping table K for this augmented mapping would be

Wall.type	Estimate.description
1234	"Metal Stud Partition Wall"
1240	"Furred Partition Wall"

Satisfaction of an augmented mapping $q'_I = q'_J$ is stricter than the non-augmented GLAV mapping $q_I = q_J$, because the condition $\pi_{schema(q_I)} q'_I(\mathbf{I}) = \pi_{schema(q_J)} q_J(\mathbf{J})$ must be met (as is the case for $q_I = q_J$), and additionally, for each $t_I \in q'_I(\mathbf{I})$, there must be a $t_J \in q'_J(\mathbf{J})$ such that $\langle t_I[k_I], t_J[k_J] \rangle \in K$ and $\pi_{schema(q_I)} t_I = \pi_{schema(q_J)} t_j$. In the following section, we describe how to use augmented mappings to find Δ_q.

C. Finding Δ_q

In order to find Δ_q, we materialize $q_I(\mathbf{I})$ and $q_J(\mathbf{J})$ as I and J respectively, perform the join $(I \bowtie K) \bowtie (K \bowtie J)$, and iterate through the result. For each tuple which has differing values (e.g. qty), we either perform an insertion (if the value from J is NULL), a deletion (if the value from I is NULL), or an update otherwise. Following the example above, suppose all walls of type 1234 are deleted, a new type of partition wall is inserted, and the result of $(I \bowtie K) \bowtie (K \bowtie J)$ is

Wall.type	I.qty	Estimate.description	J.qty
1234	NULL	"Metal Stud Partition Wall"	216
1240	130	"Furred Partition Wall"	120
1300	200	NULL	NULL

Then (informally) Δ_q is

1) Delete "Metal Stud Partition Wall"
2) Update "Furred Partition Wall" by setting qty to 130
3) Insert a tuple with description corresponding to wall type 1300 and qty 200

In this case, the Δ_J corresponding to Δ_q is easy to see. Since q_J is a select-project query, we need to ensure that our deletion and update statement contain the same selection predicate (code=9250). For the third update, we do not know the item description or rate for the tuple which should be inserted. In this case we can notify the estimator that values for these attributes should be filled in. In the case of item description, having a tuple in the mapping table K for wall type 1300 would allow that value to be used.

The techniques for computing Δ_J from Δ_q are still a work in progress. Most work on view update translation has followed the approach of Bancilhon and Spyratos [4] which makes use of a complement $\widehat{q_J}$ such that the mapping $\langle q_J, \widehat{q_J} \rangle$ is injective, and that $\widehat{q_J}(\mathbf{J}) = \widehat{q_J}(\Delta_J(\mathbf{J}))$ (called a *constant* complement). Unfortunately, select-project queries such as the q_J in Figure 3 (b) do not have a constant complement. Despite this, we may still be able to partially solve the update translation problem by suggesting potential updates, or using "unknown" values, and requiring a human expert resolve these ambiguities. In the following section, we discuss remaining challenges that must be solved in order to make data coordination practical in a wide variety of real world scenarios.

IV. Additional Research

There are a number of additional research challenges which must be addressed in order to apply the proposed techniques to a wide variety of real world data coordination problems. Section IV-A describes research into efficiently implementing data coordination using the process outlined above, while Section IV-B describes extending the mapping formalism to support a greater breadth of data coordination scenarios.

A. Executing Mappings

1) Conflicting Mappings: Our solution discussed in Section III-A computes Δ_J for each mapping constraint, and takes the union of all such Δ_J. This introduces the possibility of conflicts, for example if one mapping constraint results in deleting a tuple that another mapping constraint wishes to update. Detecting such conflicts is an important issue.

2) Performance: In order for our methods to be scalable in the number of mapping constraints, we should be considerate of the plan for materializing each of the $q^i_I(\mathbf{I})$ and $q^i_J(\mathbf{J})$, as well as the design of algorithms which find Δ_q and Δ_J.

3) Creating Mappings: Given the complexity of the relationships in heterogeneous domains, creating mappings is difficult. Previous research approached this by first establishing schema *matchings*, which may not be possible between sources with heterogeneous domains. We will focus on assisting an expert user in creating accurate mappings, e.g. by suggesting attributes to use as mapping keys.

B. Extensions

1) Mapping Cardinality: In some applications it may not be possible to create a mapping $q_I = q_J$ so that $q_I(\mathbf{I})$ only returns tuples for which there is an associated tuple in $q_J(\mathbf{J})$. The algorithm for finding Δ_q must be sensitive to the cardinality of matching between I and J, to avoid incorrect insertions or deletions. Our research must fully consider data coordination in many different cardinalities.

2) Arithmetic: In Example 1, the rate (cost per unit) of items in the Estimate relation depends on a base rate (from the CostItem relation) and the particular conditions of the design. For example, if most walls of type x are over 8 feet in height, the estimate item corresponding to walls of type x costs 20% more than the base rate due to a lower rate of productivity of the carpenter installing these walls. This mapping cannot be expressed using $q_I = q_J$, but could if we extend the formalism to allow arithmetic operations on queries. Accommodating mappings of this sort requires a great deal of work on ensuring mappings are well formed.

V. Related Work

Broadly speaking, semantic integration is partitioned into two types of applications: *data integration*, which provides uniform access to multiple data sources, and *data exchange*, which transforms data from one schema to another [5]. In both cases schema mappings are of central importance. There has been a plethora of work on representing and creating schema mappings, such as [6], [7], [8], [9].

In both data exchange and data integration, GLAV mappings are common; in data exchange, these are called source to target tuple generating dependencies. Using GLAV mappings in data integration requires answering queries using views [10]. We are currently investigating how the extensive past work on this problem, especially recent work involving dependencies (e.g. [11]) is related to the problems considered here.

Early research into Active Databases [12] allowed specification of *events* of interest (i.e. insertion/deletion/update of a tuple), and rules which describe how to respond to those events. Such rules usually contain a condition on the updated tuple, and an action which should be executed, and are often called event-condition-action (ECA) rules. For example, an insertion into the Transaction relation of a bank's database should trigger an event which causes an action that updates the account balance of the customer which made the transaction.

Hyperion [2], [13] focuses on query time integration and maintenance of data consistency in a peer setting. Kantere et al. [14] proposed the use of ECA rules for coordinating data between peers in Hyperion. For example, deleting a flight by an airline may trigger a deletion of the corresponding flight in a cooperating airline's database. A major motivation of Hyperion's mapping tables is "Mediation across multiple worlds" [2], which is the same as what we call domain heterogeneity. They are correct to point out that "In a typical integration scenario, we are often dealing with *one world*". Their approach is to manage mappings between values using mapping tables, allowing the association of *which* tuples are related, but not *how* they are related. Our approach is to make principled extensions to GLAV mappings in order to allow expressive mappings between domain heterogeneous sources.

The previous work described above operates from the perspective of *change* (i.e. how a change to data source $\mathbf{I}$ corresponds to a change to data source $\mathbf{J}$.) This is a flexible approach which is appropriate when there are 1-1 tuple correspondences between data sources, as each type of event has a corresponding type of action (an insertion in $\mathbf{I}$ corresponds to an insertion in $\mathbf{J}$, a deletion to a deletion, etc.) However, when we have relationships at the value level which involve aggregation (as is the case in heterogeneous domains), there are *many* different events which can result in the same action. Referring again to the partition wall quantity relationship of Example 2 from Section III, we would need to specify how each possible design change (e.g. insertion of a wall, deletion of a wall, change of wall type, change of wall dimensions) would affect the quantity of the corresponding estimate item. Using GLAV mappings we only need to specify the quantity of such estimate items as an aggregate query on the design.

VI. Conclusions

In this paper we have described a class of problems involving coordination of data in heterogeneous domains, which have unique properties not well addressed by existing data integration techniques. We have formalized the problem of data coordination using GLAV mappings. We have proposed augmented mappings with keys and mapping tables and given an overview of our solution to the data coordination problem using these augmented mappings. We have described future research into solving the major outstanding problems and addressing many practical concerns.

References

[1] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy, "Representing and reasoning about mappings between domain models," in *AAAI/IAAI*, 2002, pp. 80–86.

[2] A. Kementsietsidis, M. Arenas, and R. J. Miller, "Managing data mappings in the hyperion project," in *ICDE*, 2003, pp. 732–734.

[3] P. Rodríguez-Gianolli, M. Garzetti, L. Jiang, A. Kementsietsidis, I. Kiringa, M. Masud, R. J. Miller, and J. Mylopoulos, "Data sharing in the hyperion peer database system," in *VLDB*, 2005, pp. 1291–1294.

[4] F. Bancilhon and N. Spyratos, "Update semantics of relational views," *ACM Trans. Database Syst.*, vol. 6, no. 4, pp. 557–575, 1981.

[5] C. Yu and L. Popa, "Constraint-based xml query rewriting for data integration," in *SIGMOD Conference*, 2004, pp. 371–382.

[6] A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa, "Nested mappings: Schema mapping reloaded," in *VLDB*, 2006, pp. 67–78.

[7] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy, "Corpus-based schema matching," in *ICDE*, 2005, pp. 57–68.

[8] R. J. Miller, L. M. Haas, and M. A. Hernández, "Schema mapping as query discovery," in *VLDB*, 2000, pp. 77–88.

[9] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[10] A. Y. Halevy, "Answering queries using views: A survey," *The VLDB Journal*, vol. 10, no. 4, pp. 270–294, December 2001.

[11] F. N. Afrati and N. Kiourtis, "Query answering using views in the presence of dependencies," in *NTII*, 2008, pp. 8–11.

[12] U. Schreier, H. Pirahesh, R. Agrawal, and C. Mohan, "Alert: An architecture for transforming a passive dbms into an active dbms," in *VLDB*, 1991, pp. 469–478.

[13] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos, "The hyperion project: from data integration to data coordination," *SIGMOD Record*, vol. 32, no. 3, pp. 53–58, 2003.

[14] V. Kantere, I. Kiringa, and J. Mylopoulos, "Supporting distributed event-condition-action rules in a multidatabase environment," *Int. J. Cooperative Inf. Syst.*, vol. 16, no. 3/4, pp. 467–506, 2007.