Model Management Engine for Data Integration with Reverse-Engineering Support

Michael N. Gubanov #1, Philip A. Bernstein *2, Alexander Moshchuk #3

#Computer Science and Engineering, University of Washington
Box 352350, Seattle, WA 98195, USA
1mgubanov@cs.washington.edu
3anm@cs.washington.edu

*Microsoft Corporation
One Microsoft Way, Redmond, WA 98052, USA
2philbe@microsoft.com

***Abstract*— Model management is a high-level programming language designed to efficiently manipulate schemas and mappings. It is comprised of robust operators that combined in short programs can solve complex metadata-oriented problems in a compact way. For instance, countless enterprise data integration scenarios can be easily expressed in this high-level language thus saving hundreds of development man-hours.**

Here we present the first model management engine that has reverse-engineering support for data integration, which is one of the most pressing metadata-oriented problems.

It merges two schemas based on the mappings between them and allows user to correct the result keeping all the mappings in sync automatically. For user it is much more convenient than determining which mappings to correct in order to get desired result. In addition, the engine supports restructuring merging which is important when the sources are structured differently and cannot be mapped directly.

While making schema merging fully automatic is not yet possible, our work simplifies and automates this process to make it practical in complex data integration scenarios.

I. Introduction

Model management offers a set of generic operators to address a wide variety of metadata-centric problems [1] in an efficient way. The operators are functions that take and return schemas and mappings. The most popular are $Match$ and $Merge$ that compute a mapping between two schemas and merge them based on that mapping [2].

Model management operators combined in short programs offer a powerful abstraction to solve complex metadata problems quickly. One such problem is Enterprise Data Integration whose scenarios can be easily built on the top of the following short program and otherwise would have required extensive development efforts:

$$map_{12} = Match(s_1, s_2)$$
$$\langle s_3, ..\rangle = Merge(s_1, s_2, map_{12}),$$

where map_{12} is a mapping between schemas s_1 and s_2; s_3 is a merged schema. However, the following problems usually arise in practice:

- Schemas created by independent designers are often structured differently and therefore cannot be mapped directly (map_{12} does not exist).
- Despite all the progress in unsupervised schema matching [3], the automatically generated mappings almost always have errors (map_{12} is only partially correct) therefore making s_3 incorrect.

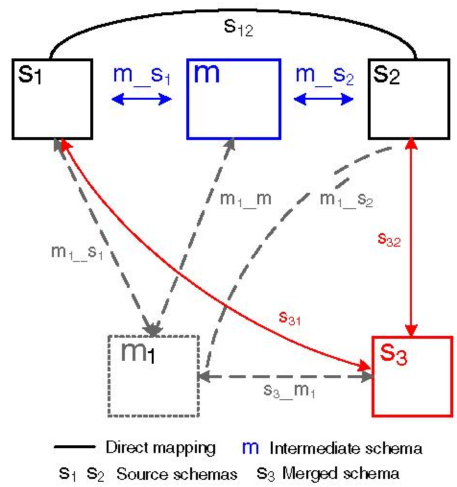

Fig. 1. Restructuring merging

In this paper, we solve both problems thereby simplifying and automating schema merging in practice. Our first contribution is an algorithm for merging schemas that cannot be mapped directly. It is implemented as a separate operator $3WayMerge$ that can be used instead of $Merge$ when map_{12} does not exist. In this case the second line in the program above would be:

$$\langle s_3, ..\rangle = 3WayMerge(s_1, s_2, m, s_{12}, m\_s_1, m\_s_2)$$

It merges two source schemas s_1, s_2 based on two mappings between them. The first one s_{12} is a partial mapping that maps the schemas' parts that can be mapped directly. The other one is a restructuring mapping that maps the schemas' parts that cannot be mapped directly. It consists of the intermediate schema m and the direct mappings m_s_1, m_s_2. Thus, by supporting two kinds of mappings $3WayMerge$ is able to merge schemas that cannot be mapped directly and cannot be merged by standard $Merge$ [2]. See Figure 1 for the illustration.

Our second contribution is a reverse-engineering algorithm that alleviates the second problem by allowing a user to make changes only to s_3 instead of solving a set of more complicated problems:

- How to change the source mappings to get desired s_3?
- Is it safe to change them and is it guaranteed that it would not corrupt correct parts of s_3?

Next, we proceed by describing a motivating scenario for restructuring merging and reverse-engineering (online at [4]).

II. Motivating Scenario

Consider a bank that would like to keep its branches synchronized with the headquarters. Headquarters stores data in a relational database which should be able to hold data from all of the provided sources. Some branches have legacy systems, but can exchange data in XML.

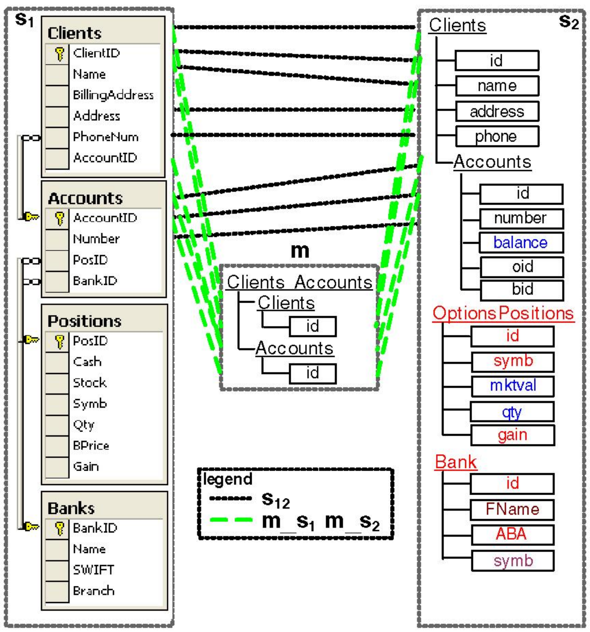

Fig. 2. Restructuring merging

Figure 2 illustrates this scenario. s_1, s_2 are branches' schemas. s_{12} is a direct mapping between s_1 and s_2. In s_1, the relationship between *Clients* and *Accounts* is of cardinality *m:1* but in s_2 it is *1:n*. Therefore neither s_1 nor s_2 by itself nor their union is able to hold the integrated data (without massive data duplication). Therefore restructuring is needed and a restructuring mapping comes into play. It is represented by schema m, direct mappings m_s_1, m_s_2, and it maps differently structured $Clients$ and $Accounts$ in s_1, s_2. The merged schema s_3 is in Figure 3 on the top. The edited, reverse-engineered schema s_3 is in Figure 3 on the bottom.

In this example, the restructuring mapping can be generated automatically by traversing the part of s_{12} that maps the primary keys of *Clients* and *Accounts* and detecting that primary key-foreign key relationships in $s_1.Clients \rightarrow s_1.Accounts$ and $s_2.Clients \leftarrow s_2.Accounts$ are in opposite directions.

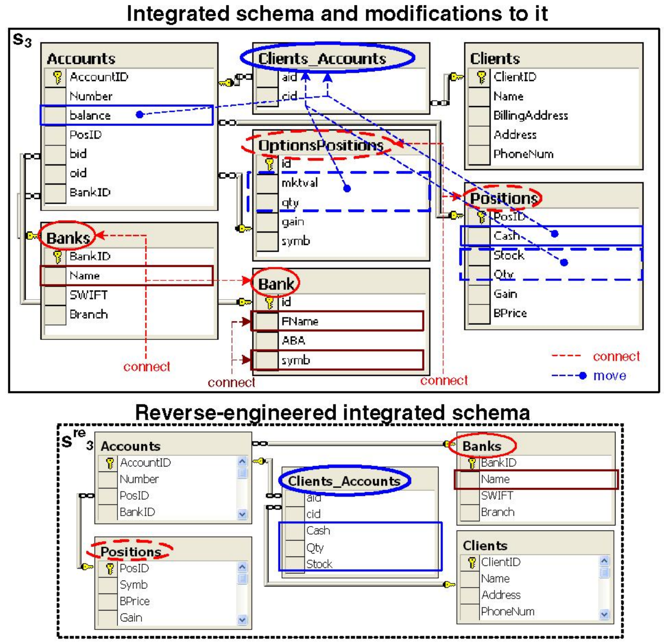

Fig. 3. Reverse-engineering

III. 3WayMerge

In this section we describe the restructuring merging algorithm. It takes two schemas, the direct mappings between them, and the restructuring mapping represented by two direct mappings and a schema; it returns the merged schema and the mappings to it.

It is easy to appreciate the idea of having two mappings. Consider what happens if one of the mappings is absent. Lack of the restructuring mapping just makes impossible merging of schemas that cannot be mapped directly. In turn, lack of the direct mapping forces everything to be mapped through a restructuring mapping. This makes the mapping almost equal to the final merge result (e.g. [5]), thus making further merging pointless and forcing user to create the final result manually.

$3WayMerge$ can be expressed in four lines of model management language. See Table I for intuition behind the operators used. More details on them are out of scope of this paper and are available in [2].Figure 1 illustrates the algorithm.

$operator$ $\boldsymbol{3WayMerge}(s_1, s_2, m, s_{12}, m\_s_1, m\_s_2)$

1) $\langle m_1, m_1\_m, m_1\_s_1\rangle = Merge(m, s_1, m\_s_1)$
2) $\langle s_3, s_3\_m_1, s_{32}\rangle =$
   $Merge(m_1, s_2, m_1\_s_1 \circ s_{12} \ \oplus m_1\_m \circ m\_s_2)$
3) $s_{31} = s_3\_m_1 \circ m_1\_s_1$
   $s_3\_m = s_3\_m_1 \circ m_1\_m \oplus s_{32} \circ \overline{m\_s_2}$

$\boldsymbol{return}\langle s_3, s_{31}, s_{32}, s_3\_m\rangle$

IV. Reverse-engineering

After $3WayMerge$ produces the merged schema s_3, the user can modify it to correct errors or to fit her needs better. These changes can be divided into two classes. The first class can be reverse-engineered to the source mappings s_{12}, m_s_1, m_s_2 so that after running $3WayMerge$ with the updated

$map = Match(s_1, s_2)$	compute a mapping between s_1 and s_2
$\langle s_3, map_{31}, map_{32}\rangle = Merge(s_1, s_2, map)$	integrate s_1 and s_2 using mapping map
$map_3 = map_1 \circ map_2$	*Compose* two direct mappings through a schema into one direct mapping
$map_3 = map_1 \oplus map_2$	*Confluence*: return a new mapping unifying mappings map_1 and map_2
$\langle s_d, s_{d\_}s\rangle = Delete(s, e)$	delete from s the elements in e
$\langle s_d, s_{d\_}s\rangle = Diff(s, map)$	return a part of s that does not participate in map
$map_i = \overline{map}$	*Invert:*swap *map's* domain and range

TABLE I

CORE MODEL MANAGEMENT OPERATORS

mappings it produces the desired modified s_3 and modified mappings to it s_{13}, s_{23}. The second class, if it exists, cannot be reverse-engineered to the source mappings and therefore needs to be processed differently. Such changes are applied directly to the modified s_3 produced after $3WayMerge$ runs on reverse-engineered mappings. This phase is called *post-processing*. There are four operations that can be applied by user to *objects* in s_3: *connect*, *split*, *delete*, and *move*. By *object* we mean an entity or attribute in this context.

$\boldsymbol{connect}(a, b)$ merges two *objects* $a, b \in s_3$ into one composite object $c \in s_3$. If a and b are entities, *connect* merges them into one entity c by taking a duplicate-free union of their attributes. Here, the attributes are considered to be identical if they are successfully matched by $Match$ operator. $\boldsymbol{split}(c)$ splits a composite object $c \in s_3$ into distinct objects $a, b \in s_3$ it consists of. $\boldsymbol{delete}(a)$ removes an object a from s_3. $\boldsymbol{move}(a, E)$ moves an attribute $a \in s_3$ to the entity $E \in s_3$.

In addition, some modifications to s_3 cannot be mapped to any modifications to the source mappings. For instance, connecting objects from the same source schema, deleting, or moving objects. Such changes are applied directly to the modified s_3 produced by a reverse-engineering iteration. This phase is called *post-processing*.

Except making changes to s_3, post-processing operations keep the *connect-*, *split-*, *delete-*, and *move-* logs, which consist of the post-processed *objects* in chronological order. Since s_3 is regenerated on every reverse-engineering iteration, these logs storing processed objects would become invalid very quickly. Therefore, we log *projections* of the processed objects to the source schemas, which always stay the same. Figures 4 illustrate how the mappings and s_3 change during *connect, move, split*, and *delete* operations.

V. CONCLUSIONS

[5] subsumes a variety of merge algorithms present in the research literature. The authors' algorithm is similar to $3WayMerge$ without a direct mapping. Lack of the direct mapping forces everything to be mapped through a restructuring mapping making it almost equal to the final merge result.

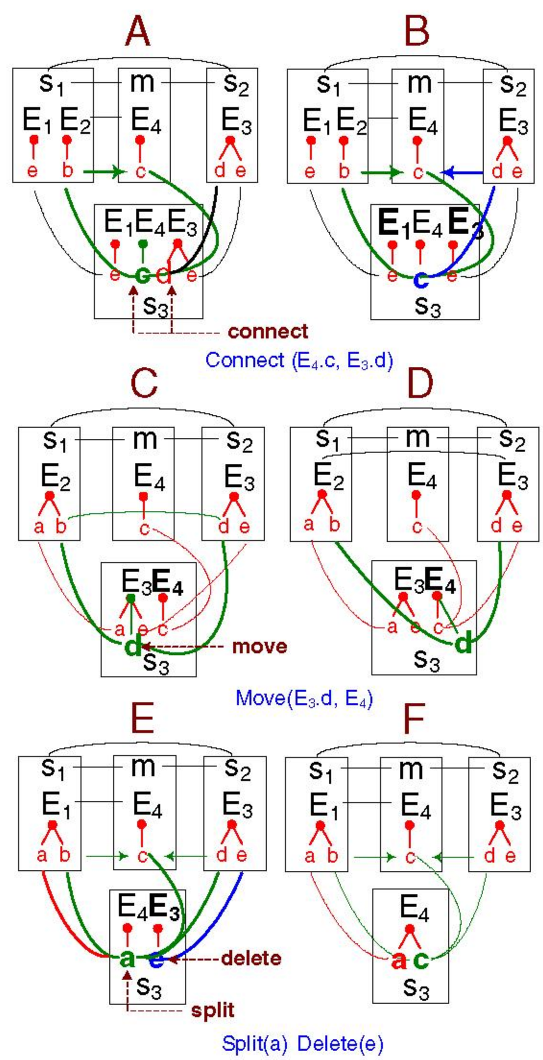

Fig. 4. Reverse-engineering and Post-processing

Our new $3WayMerge$ operator supports merging using both direct and a restructuring mapping. This makes merging functionality richer at the same time keeping the restructuring mapping small. Sometimes, all mappings can be generated fully automatically making entire schema merging fully automatic. The reverse-engineering support for $3WayMerge$ makes merging of large schemas much easier and safer.

While making schema merging fully automatic is not yet possible, our work simplifies and automates this process to make it practical in complex data integration scenarios.

REFERENCES

[1] P. A. Bernstein, "Applying model management to classical meta data problems." in *CIDR*, 2003, pp. 209–220.

[2] S. Melnik, E. Rahm, and P. A. Bernstein, "Rondo: a programming platform for generic model management," in *ACM SIGMOD*, 2003, pp. 193–204.

[3] J. Madhavan, P. A. Bernstein, K. Chen, A. Halevy, and P. Shenoy, "Corpus-based schema matching," in *IJCAI*, 2003.

[4] [Online]. Available: http://www.cs.washington.edu/homes/mgubanov

[5] R. Pottinger and P. A. Bernstein, "Merging models based on given correspondences." in *VLDB*, 2003, pp. 826–873.