

A Data Transformation Method Based On Schema Mapping

You Li⁽¹⁾, Dongbo Liu⁽²⁾, Weiming Zhang⁽¹⁾

⁽¹⁾Department of Management Science and Engineering
National University of Defense Technology, Changsha P. R. China
liyomail@sina.com

⁽²⁾College of Computer Science & Technology
Huazhong University of Science & Technology, Wuhan, P. R. China
ldb0853@sina.com

Abstract: Schema mapping is an important approach to solve the problem of data integration. This paper introduces a research prototype called SDE, which is a system for managing and facilitating the complex tasks of heterogeneous data transformation and integration. We also present a data transformation method based on schema mapping techniques. By analyzing the mappings and schemas it can automatically fulfill the task of data transformation and guarantee that the target result satisfies the target structure and constraints. It allows users to view the other participants' information as an extension of its own information system, without concerning for heterogeneity.

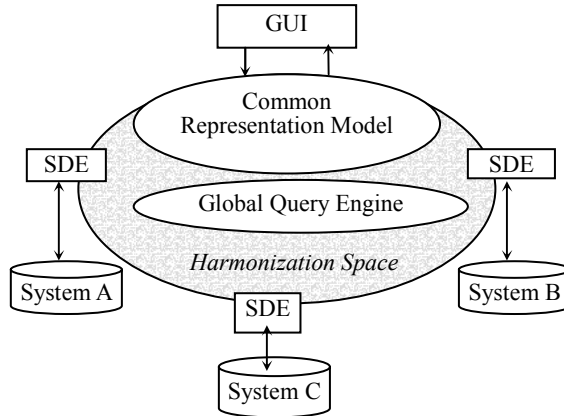
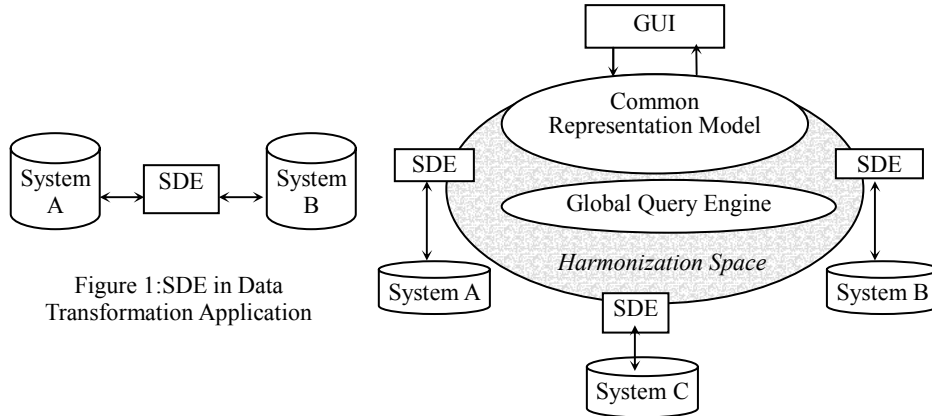
1 Introduction

Currently people have entered the age of “Information Explosion”. The emergence of the Word-Wide Web (WWW) along with the advances in data storage technology has resulted in the increments of data source in both size and quantity. These data sources are characterized in large scale, dynamic, physically distributed, autonomous and heterogeneous. Consequently we have too many “stovepipes”, which limit the interoperability. Therefore the problem of data integration and transformation has been recognized as the key factor of interoperability. Despite their importance and the wealth of research on data integration, practical integration tools are either impoverished in their capabilities or highly specialized to a limited task or integration scenario [MH+01]. As a result, integration and transformation is largely performed manually. Obviously, manually specifying data integration is a tedious, time-consuming, error-prone, and therefore expensive process. In web-based applications and services, such a manual approach is a major limitation due to the rapidly increasing number of data sources. Hence approaches for automating the data integration tasks as much as possible are badly needed to simplify and speed up the development, maintenance and use of such applications.

Schema integration and *schema mapping* are two important approaches for data integration [MH+00, HM+01]. Most work on heterogeneous data focuses on the *schema integration* problem where the target (global) schema is created from one or more source (local) schemas (and designed as a view over the sources). The target is created to reflect

the semantics of the source and has no independent semantics of its own. *Schema mapping* is used to solve the problems of data integration and transformation among independently created sources by creating mappings among schemas. It is very flexible and extensible, because the schemas may have different semantics, and may be reflected in differences in their logical structures and constraints.

SDE (Shared Data Environment) is a prototype system for managing and facilitating the complex tasks of heterogeneous data transformation and integration. It semi-automatically supports the user to fulfill the task in a faster and less labor-intensive way. Since in many cases the target schema does not depend for its definition on the identity and structure of the source, we choose the schema mapping approach in SDE. We can use SDE in both data transformation application between two heterogeneous systems (see Fig. 1) and data integration application based on common representation model (see Fig. 2). In the “Harmonization Space” in Fig. 2, all the participants are obliged to use the common representation language to cooperate with others.



The rest of this paper is divided as follows. We provide an overview of SDE in Section 2 followed by a description of the core algorithm of data transformation based on schema mapping expressions in Section 3. We briefly discuss the related work in Section 4 and conclusion in Section 5.

2 An Overview of SDE

SDE is a prototype system for managing and facilitating the complex tasks of heterogeneous data transformation and integration. It supports the generation and management of schemas, mappings between schemas, and queries between schemas. Fig. 3 highlights the main components of SDE. It consists of Gateway Generator and Local Semantic Gateway. Furthermore, Gateway Generator is composed of Schema Engine and Correspondence Engine, and Local Semantic Gateway is composed of Mapping Set and Local Query Engine. Global Query Engine is an optional component as shown in Fig. 2, which is used to deal with the global query in the integration application. Each management and reasoning component makes use of a database

management system for storing knowledge gained about schemas and integration. The following describes the functions of each component.

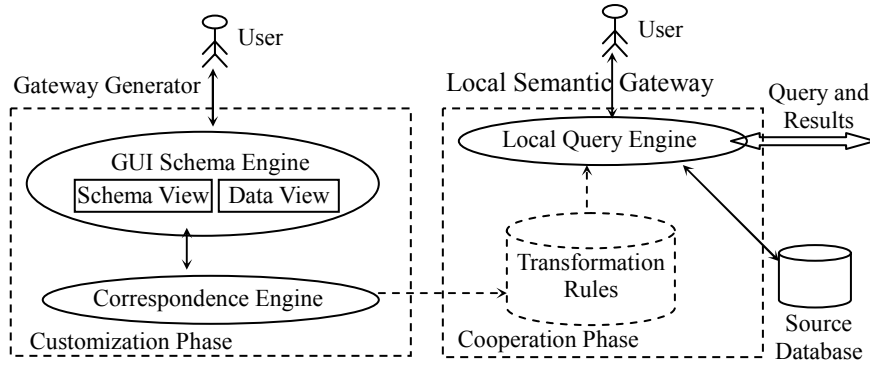


Figure 3: The Architecture of SDE

Schema Engine It makes use of a graphical user interface to show schema information and the mappings between two schemas generated by correspondence engine. The engine provides schema and data browsers to elicit and obtain feedback from users and to allow user to understand the results produced by each components.

Correspondence Engine It mines the schemas for mappings by using heuristic algorithm and machine-learning mechanism, then proposes candidate mappings to the users and finally generates the mapping set according to the user's feed back.

Local Query Engine It deals with the target queries on the target schema, automatically generates the source executable queries and finally accomplishes the task of transforming the result data into the target representation.

Global Query Engine It is an optional component and only exists in the Harmonization Space as shown in Fig. 2. According to the source descriptions stored in the repository, it decomposes the global query into several sub-queries and passes them to the local query engines. Global query engine is also responsible for combining the results and dealing with the redundancy.

Heterogeneous data transformation can be fulfilled through the Customization Phase and the Cooperation Phase. In the customization phase, the user prepares the system to be "mappable", and in the cooperation phase, the user can communicate with the other users (according to the "Local As View").

During the customization phase, the user, through a GUI in schema engine, accesses the schema information. At the same time, the correspondence engine mines the schemas for the mappings and proposes candidate mappings to the user. Lastly, according to the user's feed back, the gateway generator binds the mappings together with the local query engine, creates the local semantic gateway.

In the cooperation phase, the local semantic gateway deals with the queries on the target, then generates the source executable queries and finally fulfils the task of transforming the result data into the target representation.

SDE allows users to view the other participants' information as an extension of its own information system, without concerning for heterogeneity. By analyzing the mappings and schemas it can automatically generate the source executable query. This is distinguished from some previous methods, which usually generate the queries or global

views in advance. It also guarantees that the target result satisfies the target structure and constraints. Furthermore, when the data source changes, we need to do nothing but modify the mappings in local semantic gateway. The changes will not affect other data sources. So it is adaptive, flexible and extensible.

3 Data Transformation Algorithm in Local Query Engine

We now present the data transformation algorithm, which is used to support the local query engine. To keep the notation simple, we assume the source and target schema are represented in the relational model.

3.1 Notation

Before presenting our algorithm, we outline the notation we will be using.

We use the symbol S to denote source schema and the symbol T to denote the target schema. $A^S = \{s_1, \dots, s_p\}$ represents the set of all source attributes, where s_i is an attribute in A^S . In the same way, $A^T = \{t_1, \dots, t_q\}$ denotes the set of all target attributes. The domain of an attribute s_i (or t_i) is denoted $dom(s_i)$ (or $dom(t_i)$). We will represent the mapping set as $M = \{m_1, \dots, m_p\}$, where m_i is a mapping denoting the value correspondence between the schemas. It can be expressed as follows:

$$m_i : dom(s_1) \times dom(s_2) \dots \times dom(s_q) \rightarrow dom(t) \quad (s_i \in A^S, t \in A^T)$$

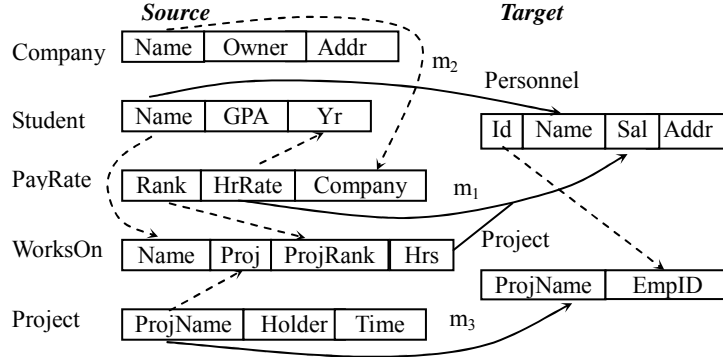


Figure 4: Example Schemas

Example 3.1 Consider the two schemas of Fig. 4. Suppose a user has indicated that the product of the values in the PayRate(HrRate) and WorksOn(Hrs) attributes should also appear in Personnel(Sal). This value correspondence is represented by the mapping m_1 . And m_2 means Student(Name) is correspond to Professor(Name).

$$m_1 : \text{PayRate}(\text{HrRate}) * \text{WorksOn}(\text{Hrs}) \rightarrow \text{Personnel}(\text{Sal})$$

$$m_2 : \text{Student}(\text{Name}) \rightarrow \text{Personnel}(\text{Name})$$

Let $SourceAttrs(M) = \{s_1, s_2, \dots, s_q\}$ be the set of all source attributes used in M , $TargetAttrs(M) = \{t_1, t_2, \dots, t_p\}$ be the set of all target attributes used in M . We use Q^S to

denote the query over source schema, and Q^T to denote the query over target schema. Their results are represented as R^S and R^T , respectively. $Attrs(Q^T)$ represents the attribute set used in Q^T .

3.2 The Core Algorithm

The heterogeneity of the schemas leads to the multiplicity of the mappings. Except for one-to-one mappings, we also have many-to-one mappings that single target attribute relates to more than one mapping, and mapping function, such as *concat()* and *multiply()*. We can manage the mapping functions by decomposing and composing their variables. By using the *grouping algorithm*, we can solve the problem of many-to-one mapping. The algorithm divides the set of mappings into subsets, which satisfy certain constraints and include only one-to-one mappings. Some of the candidate sets can be mapped into SQL queries. After executing these queries, we use the UNION operation to horizontally compose the sub-results into one integrated result. Even for the one-to-one mappings, we also need to find the way of joining the tuples. *Joining algorithm* uses the join operator to vertically compose the tuples by mining the data for possible keys and foreign keys.

We divide the algorithm into five phases including pre-processing, grouping, computing joining constraints, query execution and results combining.

Algorithm 3.1 – Main Algorithm
Input: Query on Target Schema Q^T
Set of Mappings M_0
Body: $A \leftarrow Attrs(Q^T)$
 $M \leftarrow Correspond(M_0, A)$
 $(\{M_1, M_2, \dots, M_k\}, G) \leftarrow Grouping(M)$
 $R^S \leftarrow \phi$
For each M_i
 $J_i \leftarrow Joining(M_i, G)$
 $Q_i^S \leftarrow Replace(Q^T, M_i) + J_i$
 $R_i^S \leftarrow Execute(Q_i^S)$
 $R^S = R^S \cup R_i^S$
 $Replace(R^S, M_i) = R^T$
Output: Set of Result R^T

The main task of the pre-processing phase is to find out the corresponding mapping set M according to the target query Q^T and the mapping set M_0 in the local semantic gateway.

$$M = \{ m_i \mid m_i \in M_0, TargetAttr \ s(m_i) \subset Attrs(Q^T) \}$$

As shown in Algorithm 3.1, the function $Attrs(Q^T)$ finds out the target attribute set A used in Q^T . The function $Correspond(M_0, A)$ extracts all the mappings whose target attributes are in A and constructs the corresponding set M .

Example 3.2 Consider the two schemas of Fig. 5. M_0 is the mapping set in the local semantic gateway, $M_0 = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7\}$. $C^S = \{c_1, c_2, c_3, c_4\}$ denotes the key and foreign constraints of source schema. Q^T is a query over the target.

```

 $Q^T$ : SELECT  G.t2, H.t4, H.t5
      FROM    G, H
      WHERE   G.t1=H.t3

```

The result of the pre-processing phase is $A=\{t2, t4, t5\}$, $M=\{m_1, m_2, m_3, m_4, m_5, m_6\}$.

In the grouping phase, we horizontally divide the mapping set M into subsets in order to solve the many-to-one problem. These subsets satisfy certain constraints and contain only one-to-one mappings. At the same time, we associate the source schema with digraph $G=(V, E)$. Each vertex v_i is assigned to a table in source schema. The edge set is constructed according to the tables' dependency relationship, i.e. for each pair of vertices v_i, v_j in $V(G)$, if the table v_i has a foreign key of v_j , then we create an arc $\langle v_i, v_j \rangle$.

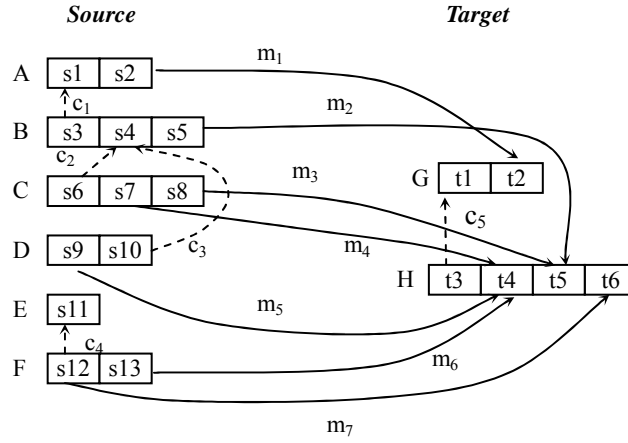


Figure 5: Example Schemas

Example 3.3 Continuing the example, The candidate set M is grouped into four subsets in the grouping phase. They are M_1, M_2, M_3 and M_4 where $M_1=\{m_1, m_4, m_3\}$, $M_2=\{m_1, m_4, m_2\}$, $M_3=\{m_1, m_5, m_2\}$, $M_4=\{m_1, m_5, m_3\}$. Fig. 6 is the digraph G constructed according to the source schema.

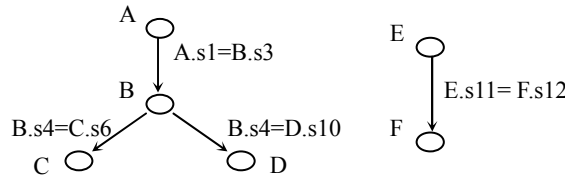


Figure 6: The digraph G

In the phase of computing joining constraints, the joining algorithm figures out joining constraint set J_i according to each M_i .

Example 3.4 Continuing the example, we can get the joining constraint set J_i of each M_i as follows:

$$\begin{aligned}
 J_1 &= \{A.s1=B.s3, C.s6=B.s4\} \\
 J_2 &= \{A.s1=B.s3, C.s6=B.s4\}
 \end{aligned}$$

$$J_3 = \{A.s1=B.s3, B.s4=D.s10\}$$

$$J_4 = \{A.s1=B.s3, D.s10=B.s4, C.s6=B.s4\}$$

In the query execution phase, the function $Replace(Q^T, M_i)$ transforms the Q^T into the source query according to each subset M_i . By adding the joining constraints J_i generated in previous phase, we get the local executable query Q_i^S . Symbol R_i^S denotes the result set of function $Execute(Q_i^S)$.

Example 3.5 In the example 3.2, for each M_i , we get the following local executable Q_i^S that transformed from the target query Q^T .

Q_1^S : SELECT A.s2, C.s7, C.s8 FROM A, B, C
WHERE (A.s1=B.s3) AND (C.s6=B.s4)

Q_2^S : SELECT A.s2, C.s7, B.s5 FROM A, B, C
WHERE (A.s1=B.s3) AND (C.s6=B.s4)

Q_3^S : SELECT A.s2, D.s9, B.s5 FROM A, B, C
WHERE (A.s1=B.s3) AND (D.s10=B.s4)

Q_4^S : SELECT A.s2, D.s9, C.s8 FROM A, B, C
WHERE (A.s1=B.s3) AND (C.s6=B.s4) AND (D.s10=B.s4)

In the last phase, we use the union operator to horizontally combine R_i^S into one integrated result R^S . Lastly, we use function $Replace()$ to transform R^S into R^T . The function $Replace()$ is also responsible for managing the mapping functions such as $concat()$, $multiply()$ by means of decomposing and composing their variables.

Example 3.6 We consider the mapping m_1 in Fig. 4. During the Q^T to Q^S transformation, the $Replace()$ function decomposes the attribute $Personnel(Sal)$ into $WorksOn.Hrs$ and $PayRate.HrRate$, and composes their results into $Personnel(Sal)$ conversely.

3.3 Grouping Algorithm

In the grouping phase, we use the grouping algorithm to horizontally decompose the mapping set M into several subsets M_i , which satisfies certain constraints and only contains one-to-one mapping. In this way, we change the problem of many-to-one into one-to-one problem.

The decomposed subsets M_i should satisfy the following constraints.

- 1) $\forall m_l, m_k \in M_i (m_l \neq m_k), TargetAttrs(m_l) \cap TargetAttrs(m_k) = \Phi$
- 2) $\forall t \in Attrs(Q^T), \exists m_k \in M_i \text{ s.t. } t \in TargetAttrs(m_k)$
- 3) $\forall v_i, v_j \in V_{Sub}^k \rightarrow Path_Exist(v_i, v_j, G_0) = True$

The first and the second constraints require that each target attribute in Q^T should relate and only relate to one mapping. We assume that V_{Sub}^k is a set of tables that contain source attributes of the decomposed subset M_k , $V_{Sub}^k \subset V(G)$. G_0 is an underlying graph of digraph G . The third constraint requires each pair of vertices should be connected, i.e. each pair of tables in V_{Sub}^k should have direct or indirect dependency relation.

Algorithm 3.2 – Grouping Algorithm

Input: Set of Mappings M

Source Schema S

Body: $A = \text{TargetAttrs}(M)$

$A^\alpha = \{t_i \mid \text{NumOfMappings}(t_i) = 1\}$

$A^\beta = A - A^\alpha$

$M^\alpha = \{m_i \mid m_i \in M, \text{TargetAttrs}(m_i) \subset A^\alpha\}$

$n_0 = \text{Num}(A^\beta)$

for each $t \in A^\beta$

$M_i^\beta = \{m_j \mid m_j \in M, t \in \text{TargetAttrs}(m_j)\} \quad (i=1, 2, \dots, n_0)$

$m_0 = \prod_{i=1}^{n_0} |M_i^\beta|$

$M' = \{ \{m_1, m_2, \dots, m_{n_0}\} \mid m_i \in M_i^\beta, i=1, 2, \dots, n_0 \} = \{M_k^\gamma \mid k=1, 2, \dots, m_0\}$

$M_k = M_k^\gamma \cup M^\alpha$

$M_{\text{group}} \leftarrow \phi$

$G = (V, E) = \text{Digraph}(S)$

$G_0 = \text{Graph}(G)$

For each M_k

$V_{\text{Sub}}^k = \text{SourceTables}(M_k)$

If for all $v_i, v_j \in V_{\text{Sub}}^k$ $\text{Path_Exist}(v_i, v_j, G_0) = \text{True}$

then $M_{\text{group}} = M_{\text{group}} \cup \{M_k\}$

Output: $M_{\text{group}} = \{M_1, M_2, \dots, M_n\}$

Diagram G

The process of grouping can be divided into four steps as follows.

Step 1: Grouping the target attributes. We extract all the target attributes of M , and construct the target attribute set A . By using the function $\text{NumOfMapping}()$ to figure out the number of mappings relates to each target attribute in A , we divide A into subsets A^α and A^β . A^α consists all the target attributes that only each one relates to one mapping. A^β consists all the target attributes that each one relates to more than one mapping.

Example 3.7 In the example of Fig. 5, we can get $A = \{t2, t4, t5\}$, $A^\alpha = \{t2\}$, $A^\beta = \{t4, t5\}$.

Step 2: Grouping mapping set. Firstly we choose the mappings whose target attributes are in A^α and construct mapping set M^α , which only contains one-to-one mappings. For each element t in A^β , we select the mappings whose target attributes are t , and construct the mapping set M_i^β , respectively.

Example 3.8 Continuing the example of Fig. 5, we get $n_0 = 2$, $M^\alpha = \{m_1\}$, $M_1^\beta = \{m_4, m_5, m_6\}$, $M_2^\beta = \{m_2, m_3\}$.

Step 3: Reconstructing the mapping set. By extracting one mapping from each M_i^β respectively, we construct M_k^γ . $M' = \{M_k^\gamma \mid k=1, 2, \dots, m_0\}$ is the set of M_k^γ . m_0 is the order of M' , and can be got by the formula $m_0 = \prod_{i=1}^{n_0} |M_i^\beta|$. Lastly, for each M_k^γ , we get M_k by

using the union operation, i.e. $M_k = M_k^\gamma \cup M^\alpha$. M_k will satisfy the constraints (1) and (2).

Example 3.9 Continuing the example, we can get $m_0 = \prod_{i=1}^2 |M_i^\beta| = 3 \times 2 = 6$. $M' = \{M_1^\gamma, M_2^\gamma, M_3^\gamma, M_4^\gamma, M_5^\gamma, M_6^\gamma\}$, $M_1 = \{m_1, m_4, m_2\}$, $M_2 = \{m_1, m_4, m_3\}$, $M_3 = \{m_1, m_5, m_2\}$, $M_4 = \{m_1, m_5, m_3\}$, $M_5 = \{m_1, m_6, m_2\}$, $M_6 = \{m_1, m_6, m_3\}$.

Step 4: Refining M_k . In order to make the output grouping set M_{group} satisfy the

constraint (3), we must refine its element M_k . Function $Digraph(S)$ is used to mine the source schema for dependency information, and associate the source schema with digraph $G=(V, E)$.

For each subset M_k , we construct the set V_{Sub}^k of tables that contains the target attribute of M_k . G_0 is an underlying graph of digraph G . In order to make M_k satisfy the third constraint, we use the function $Path_Exist(v_i, v_j, G_0)$ to figure out if there is a path between each pair of vertices in V_{Sub}^k . If there is, then $M_{group} = M_{group} \cup \{M_k\}$, otherwise eliminate M_k . Here the output mapping set M_{group} satisfies all the constraints.

Example 3.10 Continuing the example, we get the result as follows: $V_{Sub}^1 = \{A, C, B\}$, $V_{Sub}^2 = \{A, C\}$, $V_{Sub}^3 = \{A, D, B\}$, $V_{Sub}^4 = \{A, D, C\}$, $V_{Sub}^5 = \{A, F, B\}$, $V_{Sub}^6 = \{A, F, C\}$. We eliminate M_5 and M_6 because F in V_{Sub}^5 and V_{Sub}^6 are isolate. Finally we get $M_{group} = \{M_1, M_2, M_3, M_4\}$.

3.4 Joining Algorithm

In the grouping phase, we horizontally divide the set of mappings into subsets, which satisfy certain constraints and contain only one-to-one mappings. Even for the one-to-one mappings, we also need to find the way of joining the tuples. *Joining algorithm* uses the join operator to vertically combine the tuples by mining the data for possible keys and foreign keys. The main task of joining algorithm is to compute the joining constraint set J_i according to each M_i .

Algorithm 3.3 – Joining Algorithm

Input: Set of Mappings M_i
 Digraph G
Body: $V_{Sub} = \text{SourceTables}(M_i)$
 For all $v_i, v_j \in V_{Sub}$
 $Path^A = \text{AllPath}(v_i, v_j, G)$
 $V_0 = \text{Vertex}(Path^A)$
 $H = G(V_0)$
 For H
 $V_{root} = \{v_i \mid \text{id}(v_i) = 0, v_i \in V_{Sub}\}$
 $V_{other} = V_{Sub} - V_{root}$
 $E' \leftarrow \emptyset$
 For each $v_i \in V_{root}$
 For each $v_j \in V_{other}$
 $Path^S = \text{ShortPath}(v_i, v_j, Path^A)$
 $E_{path} = \text{GetEdges}(Path^S)$
 $E' = E' \cup E_{path}$
 $J_i \leftarrow \emptyset$
 For each $e_j \in E'$
 $c_j = \text{Key_constraint}(e_j, \text{meta_data})$
 $J_i = J_i \cup \{c_j\}$
Output: Joining Constraints J_i

For each mapping set M_i , Joining algorithm generates the joining constraint set J_i . The process of computing joining constraints can be divided into four steps as follows.

Step 1: Construct induced subdigraph H of digraph G . For each input mapping set M_i , we construct the set V_{Sub} of tables that contains the target attribute of M_i . Symbol $Path^A$ is used to denote all of the paths that exists between each pair of vertices in V_{Sub} . We use $Vertex()$ to extract all of the vertices in $Path^A$, and construct V_0 as the set of these vertices. H is an induced subdigraph of G , which is induced by V_0 .

Step 2: Constructing the subset of tables V_{root} and V_{other} . In subdigraph H , subset V_{root} contains the vertices of V_{Sub} , whose in-degree are equal to zero, while subset V_{other} contains all the other elements of V_{Sub} .

Step 3: Constructing edge subset E' . For each element v_i in V_{root} and each element v_j in V_{other} , we use the function $ShortPath()$ to find the shortest path from the set of path $Path^A$ and denote it as $Path^S$. By using the function $GetEdges()$ to extract all of the edges in $Path^S$, we construct the edge subset E' .

Step 4: Constructing the joining constraint set J_i . For each edge $e_j = \langle v_k, v_l \rangle$ in E' , we use $Key_constraints()$ to mine the source schema for dependency information, and generate the joining constraint c_j , which is used to construct the joining constraint set J_i . Joining constraint c_j is the key or foreign key relationship between the vertices v_k and v_l .

Example 3.11 Continuing the example, we can get table 1.

Table 1. The Values of Variables in Step 3 and Step 4

M_i	V_{Sub}	V_{root}	V_{other}	E'	J_i
M_1	{A,C,B}	{A}	{C,B}	{{(A,B), (B,C)}}	{A.s1=B.s3, C.s6=B.s4}
M_2	{A,C}	{A}	{C}	{{(A,B), (B,C)}}	{A.s1=B.s3, C.s6=B.s4}
M_3	{A,D,B}	{A}	{D,B}	{{(A,B), (B,D)}}	{A.s1=B.s3, B.s4=D.s10}
M_4	{A,D,C}	{A}	{C,D}	{{(A,B), (B,C)}, (B,D)}	{A.s1=B.s3, D.s10=B.s4, C.s6=B.s4}

We now present a typical example to illustrate our joining approach.

Example 3.12 Consider the two schemas of Fig. 4. Q^T is the query over the target.

Q^T : SELECT Personnel.Sal, Project.ProjName
FROM Personnel,Project
WHERE Personnel.ID=Project.EmplID

The following table shows the result we can get during the process.

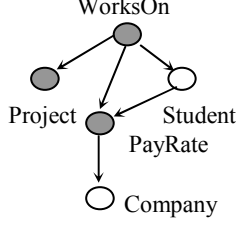
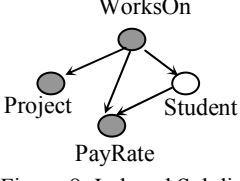
4 Related Work

We have already described the differences between classical schema integration, which is primarily a schema design problem, and the schema mapping problem we have addressed here.

The Clio tool is a collaboration between IBM Almaden Research Center and the University of Toronto [MH+01,HM+01]. It can automatically generate the source executable query according to the mapping set and the user's requirement. It is extensible and flexible. The Clio algorithm can be divided into four phases: *Group Value Correspondence*, *Select Candidate sets*, *Rank all Covers* and *Generate Query*. In the phase of Grouping, the Clio algorithm finds all the possible mapping subsets that each

subset contains *at most* one mapping per target attribute of Q . As a result, it generates lots of candidate sets. In the third phase, we attempt to find the subset Γ of the candidate sets that covers all mappings in the corresponding set M (that is, every mapping in M appears at least once in Γ). If there is more than one cover, Clío ranks them and picks out the better one, and build the query from the selected cover.

Table 2. The Values of Variables in Example 3.12

Variable	Value	Variable	Value
M_0	$\{m_1, m_2, m_3\}$	M	$\{m_1, m_3\}$
M_{group}	$\{\{m_1, m_3\}\}$	V_{sub}	$\{Project, WorksOn, PayRate\}$
$Path^A$	$\{WorksOn-Project, WorksOn-PayRate, WorksOn-Student-PayRate\}$	V_0	$\{Project, WorksOn, PayRate, Student\}$
V_{root}	$\{PayRate, Project\}$	V_{other}	$\{WorksOn\}$
$Path^S$ ($WorksOn$, $Project$)	$\{WorksOn-Project\}$	$Path^S$ ($WorksOn$, $PayRate$)	$\{WorksOn-PayRate\}$
E'	$\{(WorksOn, Project), (WorksOn, PayRate)\}$	J_1	$\{Project.ProjName=WorksOn.Proj, PayRate.Rank=WorksOn.ProjRank\}$
G	 <p>Figure 7: Digraph G</p>	H	 <p>Figure 8: Induced Subdigraph H</p>
Q^S	<pre> SELECT Project.ProjName, WorksOn.Hrs, PayRate.HrRate FROM Project, WorksOn, PayRate WHERE (Project.ProjName=WorksOn.Proj) AND (PayRate.Rank=WorksOn.ProjRank) </pre>		

Comparing with Clío, SDE algorithm finds all the possible mapping subsets that each subset contains *one and only one* mapping per target attribute of Q . As a result, it reduces the quantity of candidate sets. We finally combine all the subsets and build the query from it. For there is only one cover, the Clío's ranking phase is no longer needed. We assume n , m is the source attributes and the schema attributes of query Q , respectively, k is the quantity of the subsets generated in the grouping phase. For example, if $n=5, m=4$, the mappings between the source schema and the target schema is shown in Figure 9, then we can get k_{SDE} and $k_{Clío}$ as follows.

$$k_{SDE} = C_1^1 C_1^1 C_2^1 = 1 \times 2 = 2$$

$$k_{Clio} = C_5^1 + C_3^2 + C_3^1 C_2^1 + C_3^3 + C_3^2 C_2^1 + C_3^3 C_2^1 = 5 + 3 + 6 + 1 + 6 + 2 = 23$$

The following tables show the max number of the candidate sets generated by Clio and SDE in the case of $n=4$, $n=5$, $n=6$, respectively. We use m as the x-axis and k as the y-axis. We can see from it that the more target attributes we have in Q^T , the more efficient of our SDE.

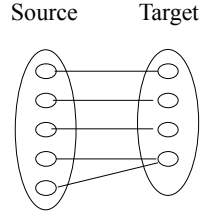


Figure 9: Example Schemas

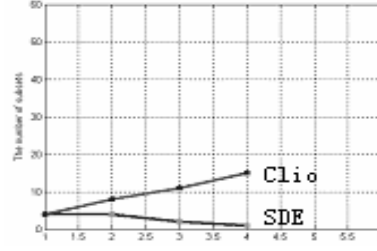


Figure 10: when $n=4$

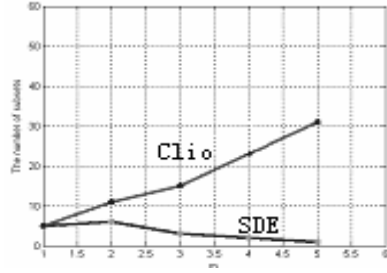


Figure 11: when $n=5$

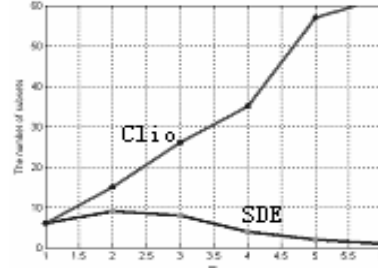


Figure 12: when $n=6$

5 Conclusions

We have discussed SDE, a system for managing and facilitating the complex tasks of heterogeneous data transformation and integration. We also present a data transformation method based on schema mapping techniques. It can automatically generate the source executable query according to the mapping set and the user's requirements. This is distinguished from some previous methods, which usually generate the queries or global views in advance. Furthermore, it is adaptive, flexible and extensible for it only needs to modify the mappings in the local semantic gateway when data source changes.

Bibliography

- [DD99] R. Domenigand, K. R. Dittrich,. An Overview and Classification of Mediated Query Systems. *In SIGMOD Record*, 28(3), 1999.
- [DD+01] AnHai Doan, P. Domingos, A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. *In SIGMOD Record*, 2001.

- [DF02] M. Dell’Erba, O. Fodor, F. Ricci, H. Werthner.: Harmonise: A Solution for Data Interoperability. *IFIP I3E* 2002.
- [DR02] Hong-Hai Do, Rehard Rahm., COMA-A System for Flexible Combination of Schema Matching Approaches. In *Proc. of the Int’l Conf. on Very Large Data Bases (VLDB)*, Hong Kong, China, 2002.
- [Le00] Alon Y. Levy. Logic-Based Techniques in data integration. In *Logic Based Artificial Intelligence*, Jack Minber(ed.). Kuwer, 2000.
- [LM+95] A.Y.Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *Proc. of the ACM Symp. On Principles of Database Systems (PODS)*, San Jose, CA, May 1995.
- [FD+02] O. Fodor, M. Dell’Erba, F. Ricci, A. Apada and H. Werthner. Conceptual Normalisation of XML Data for Interoperability in Tourism. In *Proceedings “Workshop on Knowledge Transformation for the Semantic Web (KTSW)”*, ECAI, 2002.
- [FK+03] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, pp. 207-224, Jan 2003.
- [HG+96] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breuning and V. Vassalos. Template-Based Wrappers in the TSIMMIS System. In *Proc. ACM SIGMOD Conference*, 1996.
- [HM+99] L. M. Haas, R. J. Miller, B. Niswonger, M. Tork Roth, P. M. Schwarz, and E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, 22(1):31- 36, 1999.
- [HM+01] Mauricio A. Hernández, Renée J. Miller, Laura M. Haas. Clio: A Semi-Automatic Tool For Schema Mapping. In *SIGMOD*, 2001.
- [MB+01] J.Madhavan, P.A.Bernstein, W.Rahm, Generic Schema Matching with Cupid, *VLDB* 2001.
- [MH+00] R. J. Miller, L. M. Haas, and M. Hernhdez. Schema Mapping as Query Discovery. In *Proc. of the Int’l Conf. on Very Large Data Bases (VLDB)*, pages 77-88, Cairo, Egypt, September 2000.
- [MH+01] R. J. Miller, M. A. Hernández, L. M. Haas, L. Yan, C. T. Howard Ho, R. Fagin, L. Popa. The Clio Project: Managing Heterogeneity. In *SIGMOD Record*, 30 (1), 2001.
- [PV+02] L. Popa, Y. Velegrakis, M. Hernandez, R. J. Miller, R. Fagin. Translating Web Data. In *VLDB*, pp. 598-609, Aug 2002.
- [RB+01] Erhard Rahm, Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. In *VLDB*, 10:334-350,2001.
- [RS97] M. Tork Roth, and P. Schwarz. Don’t Scrap It, Wrap It! A Wrapper Architecture for Legacy Sources. In *Proc. VLDB Conference*, 1997.