Data Integration by
Bi-Directional Schema Transformation Rules

Peter McBrien
Dept. of Computing,
Imperial College,
London SW7 2BZ
pjm@doc.ic.ac.uk

Alexandra Poulovassilis
School of Computer Science
and Information Systems,
Birkbeck College,
Univ. of London,
ap@dcs.bbk.ac.uk

Abstract

In this paper we describe a new approach to data integration which subsumes the previous approaches of **local as view (LAV)** *and* **global as view (GAV)***. Our method, which we term* **both as view (BAV)***, is based on the use of reversible schema transformation sequences. We show how LAV and GAV view definitions can be fully derived from BAV schema transformation sequences, and how BAV transformation sequences may be partially derived from LAV or GAV view definitions. We also show how BAV supports the evolution of both global and local schemas, and we discuss ongoing implementation of the BAV approach within the AutoMed project.*

1 Introduction

Data integration is a process by which several databases, which associated **local schemas**, are integrated to form a single virtual database with an associated **global schema**. Up to now, data integration approaches have been either **global as view (GAV)** or **local as view (LAV)** [12]. Examples of the GAV approach are TSIMMIS [4], InterViso [26] and Garlic [25] while examples of the LAV approach are IM [14] and Agora [15].

In this paper, we present a unifying framework for GAV and LAV, based on the use of reversible schema transformations. An important feature of our approach is that it is possible to extract a definition of the global schema as a view over the local schemas, and it is also possible to extract definitions of the local schemas as views over the global schema. Hence we term our approach **both as view (BAV)**. As we will see in this paper, one advantage of our BAV approach over GAV and LAV is that it readily supports the evolution of both global and local schemas.

Before introducing BAV, we briefly review GAV and LAV via an example which we will be referring to again throughout the paper. Figure 1 shows two local schemas S_1 and S_2, and an integration of these schemas into a global schema S_g.

Schema S_1 is a database of past and present undergraduate students at a university, and also records the member of staff who was their undergraduate tutor. The ug relation holds student records identified by the key attribute id (key attributes are shown underlined). The optional attribute left is the date the student left the university (optional attributes are suffixed with a #). The attribute sno is a foreign key indicating which member of staff in the tutor relation is the student's tutor (foreign keys are shown in italics).

Schema S_2 is a database of past and present PhD students at the university, and also records which member(s) of staff are the graduate supervisor(s) of the student. A PhD student keeps the same id if they previously studied an undergraduate course at the university.

Schema S_g is designed for use by the office handling enquiries for references about students, and needs to have within the student relation all current and past students, with the last degree that they studied, and when they left the university. The office also needs a record of the members of staff who last had contact with each student, as they will be asked to complete any requested references.

In GAV, the constructs of a global schema are described as views over the local schemas. These view definitions are used to rewrite queries over a global schema into distributed queries over the local databases. In our example, using a calculus-like nota-

S_g	student(id,name,left#,degree) monitors(*sno*,*id*) staff(sno,sname,dept#)

S_1	ug(id,name,left#,degree,*sno*) tutor(sno,sname)

S_2	phd(id,name,left#,title) supervises(*sno*,*id*) supervisor(sno,sname,dept)

Figure 1. Example global and local schemas

tion, the constructs of S_g can be defined as follows[1]:

$$\begin{array}{ll} G_1 & \mathsf{student}(\mathsf{id},\mathsf{name},\mathsf{left},\mathsf{degree}) = \{x,y,z,w \mid \\ & \quad \langle x,y,z,w,\_\rangle \in \mathsf{ug} \wedge \langle x,\_,\_,\_\rangle \notin \mathsf{phd} \\ & \quad \vee \langle x,y,z,\_\rangle \in \mathsf{phd} \wedge w = \text{`}\mathsf{phd}\text{'}\} \\ G_2 & \mathsf{monitors}(\mathsf{sno},\mathsf{id}) = \{x,y \mid \\ & \quad \langle x,\_,\_,\_,y\rangle \in \mathsf{ug} \wedge \langle x,\_,\_,\_\rangle \notin \mathsf{phd} \\ & \quad \vee \langle x,y\rangle \in \mathsf{supervises}\} \\ G_3 & \mathsf{staff}(\mathsf{sno},\mathsf{sname},\mathsf{dept}) = \{x,y,z \mid \\ & \quad \langle x,y\rangle \in \mathsf{tutor} \wedge \langle x,\_,\_\rangle \notin \mathsf{supervisor} \\ & \quad \vee \langle x,y,z\rangle \in \mathsf{supervisor}\} \end{array}$$

In LAV, the constructs of the local schemas are defined as views over the global schema, and processing queries over the global schema involves rewriting queries using views [13]. In our example, schema S_1 can be defined by derivation rules L_1 and L_2 below, and schema S_2 by derivation rules L_3, L_4 and L_5:

$$\begin{array}{ll} L_1 & \mathsf{tutor}(\mathsf{sno},\mathsf{sname}) = \{x,y \mid \\ & \quad \langle x,y,\_\rangle \in \mathsf{staff} \wedge \langle x,z\rangle \in \mathsf{monitors} \\ & \quad \wedge \langle z,\_,\_,w\rangle \in \mathsf{student} \wedge w \neq \text{`}\mathsf{phd}\text{'}\} \\ L_2 & \mathsf{ug}(\mathsf{id},\mathsf{name},\mathsf{left},\mathsf{degree},\mathsf{sno}) = \{x,y,z,w,v \mid \\ & \quad \langle x,y,z,w\rangle \in \mathsf{student} \wedge \langle v,x\rangle \in \mathsf{monitors} \\ & \quad \wedge w \neq \text{`}\mathsf{phd}\text{'}\} \\ L_3 & \mathsf{phd}(\mathsf{id},\mathsf{name},\mathsf{left},\mathsf{title}) = \{x,y,z,w \mid \\ & \quad \langle x,y,z,v\rangle \in \mathsf{student} \wedge v = \text{`}\mathsf{phd}\text{'} \\ & \quad \wedge w = \mathit{null}\} \\ L_4 & \mathsf{supervises}(\mathsf{sno},\mathsf{id}) = \{x,y \mid \\ & \quad \langle x,y\rangle \in \mathsf{monitors} \wedge \langle x,\_,\_,z\rangle \in \mathsf{student} \\ & \quad \wedge z = \text{`}\mathsf{phd}\text{'}\} \\ L_5 & \mathsf{supervisor}(\mathsf{sno},\mathsf{sname},\mathsf{dept}) = \{x,y,z \mid \\ & \quad \langle x,y,z\rangle \in \mathsf{staff} \wedge \langle x,w\rangle \in \mathsf{monitors} \\ & \quad \wedge \langle w,\_,\_,v\rangle \in \mathsf{student} \wedge v = \text{`}\mathsf{phd}\text{'}\} \end{array}$$

[1]We use a calculus-like notation for our examples in this paper because this is syntactically close to our actual intermediate query language, which is a comprehensions-based functional query language [21]. Such languages subsume high-level query languages such as SQL and OQL in expressiveness [3, 7].

In rule L_3, note the use of *null* for the value of the title, since no information can be derived from S_g regarding the topic title of a phd student's thesis. We will later give a more precise method of reasoning about such information, using the distinguished constant *void* to differentiate information that is not derivable from information that is actually recorded as *null*.

Note that rule L_1 does *not* define the complete extent of the relation tutor in S_1, but just those tuples of tutor which can be derived from S_g. In particular, any staff member who supervises PhD students and tutors undergraduates will not appear in this view. This is a subtle aspect of the LAV approach, since in general it is not possible to take any S_g and provide views that can identify the source of all of the data. In the terminology of [11], this means L_1 is a **complete** LAV rule. A similar argument holds for L_2. Both L_1 and L_2 could become **sound** LAV rules by the removal of the $w \neq$ 'phd' term, since they would then derive a superset of the source relations. Rules L_3, L_4 and L_5 are **exact** LAV rules, since they exactly derive the source relations from the global schema. In this particular example there is no possible exact LAV rule for L_1, as there is no way of distinguishing those members of staff who supervise PhD students and undergraduates from those staff who supervise PhD students only. A similar argument for holds for why no exact definition of L_2 is possible. A similar categorisation into sound, complete and exact is possible for GAV rules, but in practice it is usually assumed that the rules are exact in GAV.

A variation of LAV called GLAV [8] allows for the head of the view definition rules in GLAV to contain conjunctions of source relations as opposed to the single relations allowed by LAV, and to contain variables that do not appear in the body of the rule. This would be required if S_2 had defined supervises(*sno*,*title*), where a GLAV rule could provide a definition phd(id, name, left, title) $\wedge$ supervises(sno, title) without providing values to the title attribute. By contrast, a LAV approach would require that title appear in the global schema merely to allow the definition of this join.[2]

The principal disadvantage of GAV is that it does not readily support the evolution of the local schemas. For example, adding an age attribute to the phd relation in S_2 would invalidate view definitions G_1 and G_2 above. Since this maintenance problem worsens as the number of source databases increases, GAV has been

[2]If we do not allow the use of *null* values, then L_3 would not be valid. In that case a LAV rule defining phd would force the title attribute of S_2 to appear in S_g, regardless as to whether it is otherwise required to be in the global schema. However, removing the term $w = \mathit{null}$ from L_3 would result in a valid GLAV rule.

criticised as not scaling well.

LAV isolates changes to local schemas to impact only on the derivation rules defined for that schema. Hence, adding the extra age attribute to phd would only effect rule L_3 above. However, LAV has problems if one needs to change the global schema, since all the rules for defining local schemas as views of the global schema will need to be reviewed.

The remainder of this paper discusses how our BAV method may be used as a unifying framework for LAV and GAV, and how several practical advantages result. In Section 2, we present our BAV integration method. In Section 3, we show how BAV can capture all the semantic information that is present in LAV or GAV derivation rules. In particular, we show how to derive exact or complete LAV or GAV rules from BAV rules. Section 4 discusses how BAV supports the evolution of both global and local schemas. Section 5 discusses implementation of the BAV approach within the ongoing AutoMed project. Section 6 gives our concluding remarks and directions of further work.

2 The BAV Integration Method

In previous work [22, 17] we have developed a general framework to support schema transformation and integration in heterogeneous database architectures. The framework consists of a low-level **hypergraph-based data model (HDM)** and a set of primitive schema transformations defined for this model. Higher-level data models and primitive schema transformations for them are defined in terms of this lower-level common data model.

In our framework, schemas are incrementally transformed by applying to them a sequence of primitive transformation steps $t_1, \ldots, t_n$. Each primitive transformation t_i makes a 'delta' change to the schema, adding, deleting or renaming just one schema construct. Each add or delete step is accompanied by a query specifying the extent of the new or deleted construct in terms of the rest of the constructs in the schema.

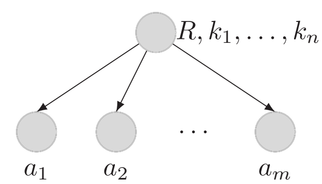
In [16, 22] we showed how our primitive transformations are rich enough to express all of the transformations commonly used in the integration of ER schemas, and we formally derived precisely what, if any, constraints on the actual data sources these transformations are conditional upon.

One advantage of using a low-level common data model such as the HDM is that semantic mismatches between modelling constructs are avoided. Another advantage is that it provides a unifying semantics for higher-level modelling constructs. In particular, in [18] we described how our framework can be applied to different high-level modelling languages such as relational, ER and UML. In [19] we showed how XML data sources can also be handled within our framework.

2.1 A simple relational data model

For our purposes in the present paper, we assume that all schemas are specified in the very simple relational data model we define below. But our discussion here is equally applicable to any data modelling language supported by our framework.

In our simple relational model, schemas are constructed from primary key attributes, non-primary key attributes, and the relationships between them. The underlying graph representation of a relation R with primary key attributes $k_1, \ldots, k_n$ and other attributes $a_1, \ldots, a_m$ is:



Thus the relation staff of S_g is modelled as by the Rel scheme $\langle\langle$staff,sno$\rangle\rangle$ with unary-tuple instances (since in this case $n = 1$), together with two Att schemes $\langle\langle$staff,sname$\rangle\rangle$ and $\langle\langle$staff,dept$\rangle\rangle$ with binary-tuple instances. Thus we write $x \in \langle\langle$staff,sno$\rangle\rangle$ to find instances of the key, and $\langle x, y\rangle \in \langle\langle$staff,sname$\rangle\rangle$ and $\langle x, y\rangle \in \langle\langle$staff,dept$\rangle\rangle$ to find values of the non-key attributes for this key.

The set of primitive transformations for schemas expressed in this data model is as follows:

- addRel($\langle\langle R, k_1, \ldots, k_n\rangle\rangle, q$) adds to the schema a new relation R with primary key attribute(s) $k_1, \ldots, k_n$, $n \geq 1$.

  The query q specifies the set of primary key values in the extent of R in terms of the already existing schema constructs (this will be a set of n-tuples).

- addAtt($\langle\langle R,a\rangle\rangle$,$c$,$q$) adds to the schema a non-primary key attribute a for relation R. The parameter c can be either null or notnull.

  The query q specifies the extent of the binary relationship between the primary key attribute(s) of R and this new attribute a in terms of the already existing schema constructs (this extent will be a set of pairs).

- delRel($\langle\langle R$,$k_1, \ldots, k_n\rangle\rangle$,$q$) deletes from the schema the relation R with primary key attribute(s)

$k_1, \ldots, k_n$ (provided all its non-primary key attributes have first been deleted).

The query q specifies how the set of primary key values in the extent of R can be restored from the remaining schema constructs.

- delAtt($\langle\langle R,a \rangle\rangle$,$c$,$q$) deletes from the schema the non-primary key attribute a of relation R.

  The query q specifies how the extent of the binary relationship between the primary key attribute(s) of R and a can be restored from the remaining schema constructs.

All primitive transformation rules have a optional additional argument which specifies a constraint on the data that must hold if the transformation is to apply. In the case of our relational model, this can be used to enforce any foreign key constraints that are required to hold. Each of the above primitive transformations, t, has an automatically derivable reverse transformation, $\bar{t}$, defined as follows:

$t : S_x \rightarrow S_y$	$\bar{t} : S_y \rightarrow S_x$
addRel($\langle\langle R, \vec{k_n} \rangle\rangle, q$)	delRel($\langle\langle R, \vec{k_n} \rangle\rangle, q$)
addAtt($\langle\langle R, a \rangle\rangle, c, q$)	delAtt($\langle\langle R, a \rangle\rangle, c, q$)
delRel($\langle\langle R, \vec{k_n} \rangle\rangle, q$)	addRel($\langle\langle R, \vec{k_n} \rangle\rangle, q$)
delAtt($\langle\langle R, a \rangle\rangle, c, q$)	addAtt($\langle\langle R, a \rangle\rangle, c, q$)

Here, the notation $\vec{k_n}$ is an abbreviation for the sequence of attributes $k_1, \ldots, k_n$, and we will use similar abbreviations for sequences of attributes in the remainder of the paper.

In [17] we show how this reversibility of schema transformations allows automatic query translation between schemas. In particular, for the simple relational data model above, if a schema S is transformed to a schema S' by a single primitive transformation step, the only cases that need to be considered in order to translate a query Q posed on S to a query Q' posed on S' are if the transformation step is a delRel or delAtt, in which case occurrences in Q of the deleted construct need to be substituted by the query q specified in the transformation. For sequences of primitive transformations, these substitutions are successively applied in order to obtain the final translated query Q'.

This translation scheme can be applied to each of the constructs of a global schema in order to obtain the derivation of each construct from a set of local schemas. These derivations can then be substituted into any query over the global schema in order to obtain an equivalent query distributed over the local schemas [10], as in the GAV approach.

The above translation scheme can equivalently be applied to each of the constructs of a local schema in order to obtain its derivation from a global schema, as in the LAV approach.

2.2 Integrating S_1 and S_2 into S_g

Using the above set of primitive transformations on our simple relational data model, Table 1 lists the transformation steps necessary to integrate the local schemas S_1 and S_2 into the global schema S_g.

An implicit first step in this integration process is to combine the local schemas into a single schema $S_1 \cup S_2$ whose constructs are precisely those of S_1 and S_2. In our particular example, there are no commonly-named relations in S_1, S_2 and S_g, and so for simplicity we just use the relation names directly in Table 1. In general, each construct from a local schema would be tagged with a unique schema identifier, as would the constructs in the target global schema. Note, if required, this implicit step can be formalised in our approach using a series of extend steps (see below) to add to S_2 the constructs of S_1, and to add to S_1 the constructs of S_2.

Table 1 uses two more primitive transformations to the ones we defined in Section 2.1: conRel and conAtt. These behave in the same way as delRel and delAtt except that they indicate that their accompanying query q may only *partially* restore the extent of the deleted construct; hence we may say that they are **complete** in the same sense as LAV/GAV rules are complete. These two contract (abbreviated as con) transformations have corresponding reverse transformations extRel and extAtt respectively, which behave in the same way as addRel and addAtt except that they indicate their accompanying query q may only *partially* construct the extent of the new construct. For these contract and extend (abbreviated as ext) transformations, the query q may be just the distinguished constant *void*, which indicates that there is no information about how to derive the extent of the deleted/new construct from the rest of the schema constructs, even partially.

This distinction between partial and complete derivations allows a precise statement to be made of the relationship between the source and target schemas, since it provides a means of stating how much information is known about a particular schema construct. To illustrate, suppose that we wanted to add a relation cs(<u>id</u>) containing all students that study in the computer science department, and that we knew that all students with degree='G500' study computer science, as do some students with degree='GG15' (and that no other students study computer science):

1. $\mathsf{addRel}(\langle\langle \mathsf{cs,id}\rangle\rangle,\ \{x \mid \langle x,y\rangle \in \langle\langle \mathsf{ug,degree}\rangle\rangle \wedge y = \text{‘G500’}\})$ would be incorrect, since it states that cs contains *just* those students on G500, and no other students.

2. $\mathsf{extRel}(\langle\langle \mathsf{cs,id}\rangle\rangle,\ \{x \mid \langle x,y\rangle \in \langle\langle \mathsf{ug,degree}\rangle\rangle \wedge y = \text{‘G500’}\})$ would be correct, since it states that cs contains those students on G500, plus some others which we have no way of determining.

3. $\mathsf{extRel}(\langle\langle \mathsf{cs,id}\rangle\rangle, \mathit{void})$ would be incorrect, since it states that we have no way of determining any of the students that belong to cs, when in fact we know that all G500 students belong to cs.

If instead it was the case that only students with G500 study computer science, then (1) would be correct. Alternatively, if instead some G500 and some GG15 students study computer science, then (3) would be correct. To make this incompleteness of knowledge explicit in query processing, an annotation *partial* can be attached to a query or sub-query to indicate that it returns a partial result. Thus $\{x \mid \langle x,y\rangle \in \langle\langle \mathsf{ug,degree}\rangle\rangle \wedge y = \text{‘G500’}\}^{partial}$ means that $\{x \mid \langle x,y\rangle \in \langle\langle \mathsf{ug,degree}\rangle\rangle \wedge y = \text{‘G500’}\}$ returns instances which may be only a subset of the complete answer.

Returning now to Table 1, we see that the transformation steps ① to ⑧ incrementally define the constructs of S_g from the constructs of S_1 and S_2. This can be regarded as the GAV aspect of our framework.

Steps ⑨ to ㉓ then incrementally remove the constructs of S_1 and S_2 from this intermediate schema, finally leaving only the constructs of S_g, as desired. The queries accompanying steps ⑨ to ⑮ show how the constructs of S_1 can be restored from those of S_g, and the queries accompanying steps ⑯ to ㉓ show how the constructs of S_2 can be restored from S_g. This can be regarded as the LAV aspect of our framework.

Generally, this is the form that schema transformation/integration takes in our framework, namely a growing phase in which new schema constructs are added, and a shrinking phase in which redundant constructs are deleted.

Note that ⑧ has the optional constraint allowed in our transformation rules, which enforces the foreign keys of the monitors relation in S_g. Similarly ⑯ has a constraint to enforce the foreign key constraints of the supervises table.

3 Correspondence between BAV and GAV/LAV

In this section we examine in more detail the correspondence between BAV and GAV, and between BAV and LAV. We first show how a GAV or LAV definition can be converted into a partial BAV definition. We then show how a complete GAV or LAV definition can be derived from a BAV definition.

BAV thus combines the benefits of both GAV and LAV in the sense that any reasoning or processing which is possible with the view definitions of GAV or LAV will also be possible with the BAV definition.

Given its expressiveness, it could be argued that data integration using the BAV approach is more complex than with GAV or LAV. However, by specifying well-known schema equivalences as BAV transformation macros the production of BAV definitions is greatly simplified, and we conclude this section with a discussion of this issue.

In what follows, we will need to use the fact that a relation R with primary key attributes $k_1, \ldots, k_n$ and other attributes $a_1, \ldots, a_m$, has a lossless-join decomposition into m relations, each containing key attributes $k_1, \ldots, k_n$ and one of the other attributes $a_1, \ldots, a_m$. Hence we can rewrite any derivation rule of the form

$$R(\vec{k_n}, \vec{a_m}) = \{\vec{x_n}, \vec{y_m} \mid E\}$$

as $m+1$ equivalent derivation rules:

$$R(\vec{k_n}) = \{\vec{x_n} \mid E\}$$
$$R(\vec{k_n}, a_1) = \{\vec{x_n}, y_1 \mid E\}$$
$$\vdots$$
$$R(\vec{k_n}, a_m) = \{\vec{x_n}, y_m \mid E\}$$

(Note that some sub-expressions of the expression E may be logically redundant for some of the above derivation rules, and can be automatically removed by standard query simplification techniques.) The original derivation rule for the relation R can thus be represented by the following sequence of primitive transformation steps, where the function $D(E)$ rewrites the expression E so that any relations it references are decomposed in the same way as R:

$$\mathsf{addRel}(\langle\langle R, \vec{k_n}\rangle\rangle, \{\vec{x_n} \mid D(E)\})$$
$$\mathsf{addAtt}(\langle\langle R, a_1\rangle\rangle, \{\vec{x_n}, y_1 \mid D(E)\})$$
$$\vdots$$
$$\mathsf{addAtt}(\langle\langle R, a_m\rangle\rangle, \{\vec{x_n}, y_m \mid D(E)\})$$

In what follows, we will refer to this equivalence between derivation rules for relations and sequences of

① $\mathsf{addRel}(\langle\langle\mathsf{student,id}\rangle\rangle,\ \{x \mid x \in \langle\langle\mathsf{ug,id}\rangle\rangle \lor x \in \langle\langle\mathsf{phd,id}\rangle\rangle\})$
② $\mathsf{addAtt}(\langle\langle\mathsf{student,name}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{ug,name}\rangle\rangle \land x \notin \langle\langle\mathsf{phd,id}\rangle\rangle$
$\qquad \lor \langle x,y\rangle \in \langle\langle\mathsf{phd,name}\rangle\rangle\})$
③ $\mathsf{addAtt}(\langle\langle\mathsf{student,left}\rangle\rangle,\mathsf{null},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{ug,left}\rangle\rangle \land x \notin \langle\langle\mathsf{phd,id}\rangle\rangle \lor \langle x,y\rangle \in \langle\langle\mathsf{phd,left}\rangle\rangle\})$
④ $\mathsf{addAtt}(\langle\langle\mathsf{student,degree}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{ug,degree}\rangle\rangle \land x \notin \langle\langle\mathsf{phd,id}\rangle\rangle$
$\qquad \lor x \in \langle\langle\mathsf{phd,id}\rangle\rangle \land y = \text{`phd'}\})$
⑤ $\mathsf{addRel}(\langle\langle\mathsf{staff,sno}\rangle\rangle,\ \{x \mid x \in \langle\langle\mathsf{tutor,sno}\rangle\rangle \lor x \in \langle\langle\mathsf{supervisor,sno}\rangle\rangle\})$
⑥ $\mathsf{addAtt}(\langle\langle\mathsf{staff,sname}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{tutor,sname}\rangle\rangle \land x \notin \langle\langle\mathsf{supervisor,sno}\rangle\rangle$
$\qquad \lor \langle x,y\rangle \in \langle\langle\mathsf{supervisor,sname}\rangle\rangle\})$
⑦ $\mathsf{addAtt}(\langle\langle\mathsf{staff,dept}\rangle\rangle,\mathsf{null},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{supervisor,dept}\rangle\rangle$
$\qquad \lor x \in \langle\langle\mathsf{tutor}\rangle\rangle \land x \notin \langle\langle\mathsf{supervisor,sno}\rangle\rangle \land y = \mathit{null}\})$
⑧ $\mathsf{addRel}(\langle\langle\mathsf{monitors,sno,id}\rangle\rangle,\ \{x,y \mid \langle y,x\rangle \in \langle\langle\mathsf{ug,sno}\rangle\rangle \land y \notin \langle\langle\mathsf{phd,id}\rangle\rangle \lor \langle x,y\rangle \in \langle\langle\mathsf{supervises,sno,id}\rangle\rangle\},$
$\qquad \langle x,y\rangle \in \langle\langle\mathsf{monitors,sno,id}\rangle\rangle \to x \in \langle\langle\mathsf{staff,sno}\rangle\rangle \land y \in \langle\langle\mathsf{student,id}\rangle\rangle)$
⑨ $\mathsf{conAtt}(\langle\langle\mathsf{tutor,sname}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{staff,sname}\rangle\rangle \land x \in \langle\langle\mathsf{tutor,sno}\rangle\rangle\})$
⑩ $\mathsf{conRel}(\langle\langle\mathsf{tutor,sno}\rangle\rangle,\ \{x \mid x \in \langle\langle\mathsf{staff,sno}\rangle\rangle \land \langle y,x\rangle \in \langle\langle\mathsf{ug,sno}\rangle\rangle\})$
⑪ $\mathsf{conAtt}(\langle\langle\mathsf{ug,name}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{student,name}\rangle\rangle \land x \in \langle\langle\mathsf{ug,id}\rangle\rangle\})$
⑫ $\mathsf{conAtt}(\langle\langle\mathsf{ug,left}\rangle\rangle,\mathsf{null},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{student,left}\rangle\rangle \land x \in \langle\langle\mathsf{ug,id}\rangle\rangle\})$
⑬ $\mathsf{conAtt}(\langle\langle\mathsf{ug,degree}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{student,degree}\rangle\rangle \land x \in \langle\langle\mathsf{ug,id}\rangle\rangle\})$
⑭ $\mathsf{conAtt}(\langle\langle\mathsf{ug,sno}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle y,x\rangle \in \langle\langle\mathsf{monitors,sno.id}\rangle\rangle \land x \in \langle\langle\mathsf{ug,id}\rangle\rangle\})$
⑮ $\mathsf{conRel}(\langle\langle\mathsf{ug,id}\rangle\rangle,\ \{x \mid x \in \langle\langle\mathsf{student,id}\rangle\rangle \land \langle x,\text{`phd'}\rangle \notin \langle\langle\mathsf{student,degree}\rangle\rangle\})$
⑯ $\mathsf{delRel}(\langle\langle\mathsf{supervises,sno,id}\rangle\rangle,\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{monitors,sno,id}\rangle\rangle \land y \in \langle\langle\mathsf{phd,id}\rangle\rangle\},$
$\qquad \langle x,y\rangle \in \langle\langle\mathsf{supervises,sno,id}\rangle\rangle \to x \in \langle\langle\mathsf{supervisor,sno}\rangle\rangle \land y \in \langle\langle\mathsf{phd,id}\rangle\rangle)$
⑰ $\mathsf{delAtt}(\langle\langle\mathsf{supervisor,sname}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{staff,sname}\rangle\rangle \land x \in \langle\langle\mathsf{supervisor,sno}\rangle\rangle\})$
⑱ $\mathsf{delAtt}(\langle\langle\mathsf{supervisor,dept}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{staff,dept}\rangle\rangle \land x \in \langle\langle\mathsf{supervisor,sno}\rangle\rangle\})$
⑲ $\mathsf{delRel}(\langle\langle\mathsf{supervisor,sno}\rangle\rangle,\ \{x \mid x \in \langle\langle\mathsf{staff,sno}\rangle\rangle \land \langle x,y\rangle \in \langle\langle\mathsf{staff,dept}\rangle\rangle \land y \neq \mathit{null}\})$
⑳ $\mathsf{delAtt}(\langle\langle\mathsf{phd,name}\rangle\rangle,\mathsf{notnull},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{student,name}\rangle\rangle \land x \in \langle\langle\mathsf{phd,id}\rangle\rangle\})$
㉑ $\mathsf{delAtt}(\langle\langle\mathsf{phd,left}\rangle\rangle,\mathsf{null},\ \{x,y \mid \langle x,y\rangle \in \langle\langle\mathsf{student,left}\rangle\rangle \land x \in \langle\langle\mathsf{phd,id}\rangle\rangle\})$
㉒ $\mathsf{conAtt}(\langle\langle\mathsf{phd,title}\rangle\rangle,\mathsf{notnull},\ \mathit{void})$
㉓ $\mathsf{delRel}(\langle\langle\mathsf{phd,id}\rangle\rangle,\ \{x \mid x \in \langle\langle\mathsf{student,id}\rangle\rangle \land \langle x,\text{`phd'}\rangle \in \langle\langle\mathsf{student,degree}\rangle\rangle\})$

Table 1. A complete BAV speciﬁcation of the integration of S_1 and S_2 into S_g

primitive transformation steps as the **decomposition rule**.

3.1 Deriving BAV from GAV

A GAV definition can be used to derive *some* of the information present in the BAV definition, as follows:

First, each GAV derivation rule has the decomposition rule applied to it. Thus, in our running example, rule G_1 generates steps ① –④ , G_2 generates ⑧ , and G_3 generates ⑤ –⑦ .

Next, each construct c of type T in the source schema is removed using a transformation step of the form con$T(c,void)$. This indicates that the construct is being removed, but (as a consequence of GAV) there is an absence of information about how to restore its extent. In our running example, this means that steps ⑨ –㉓ are partially derived as:

$\boxed{9}$ $\mathsf{conAtt}(\langle\langle\mathsf{tutor,sname}\rangle\rangle,\mathsf{notnull},\mathit{void})$
⋮
㉒ $\mathsf{conAtt}(\langle\langle\mathsf{phd,title}\rangle\rangle,\mathsf{notnull},\ \mathit{void})$
$\boxed{23}$ $\mathsf{conRel}(\langle\langle\mathsf{phd,id}\rangle\rangle,\mathit{void})$

The box placed around the circle of a transformation step indicates that the same construct is derived as in Table 1, but the query only returns a partial result compared to that in Table 1. For example, $\boxed{9}$ has a void query, as so clearly returns a partial result compared ⑨ . By contrast, ㉒ is completely defined by virtue of the fact that it was defined with a *void* query in the complete BAV integration, and hence no box is placed around the circle.

3.2 Deriving BAV from LAV

A LAV definition can also be used to derive *some* of the information present in the BAV definition:

Starting with the set of LAV rules, the decomposition rule is applied to each of them. In our running example, LAV rules L_1 to L_5 generate ㉓–❾, the reverse transformation steps of ⑨ –㉓ . However, in this case, *all* the BAV transformation steps generated must be extend rather than add ones, since with LAV it may be the case that not all the information in a local construct is derivable from the global schema:

㉓ $\mathsf{extRel}(\langle\langle \mathsf{phd,id}\rangle\rangle, \{x \mid x \in \langle\langle \mathsf{student,id}\rangle\rangle$
$\quad \vee \langle x, \text{‘phd’}\rangle \in \langle\langle \mathsf{student,degree}\rangle\rangle\})$

⋮

❿ $\mathsf{extRel}(\langle\langle \mathsf{tutor,sno}\rangle\rangle,$
$\quad \{x \mid x \in \langle\langle \mathsf{staff,sno}\rangle\rangle \wedge \langle y, x\rangle \in \langle\langle \mathsf{ug,sno}\rangle\rangle\})$

❾ $\mathsf{extAtt}(\langle\langle \mathsf{tutor,sname}\rangle\rangle, \mathsf{notnull}, \{x, y \mid$
$\quad \langle x, y\rangle \in \langle\langle \mathsf{staff,sname}\rangle\rangle \wedge x \in \langle\langle \mathsf{tutor,sno}\rangle\rangle\})$

Note that in this case ❿ and ❾ remain unboxed, by virtue of the fact that the inversion of contract transformations ⑩ and ⑨ are the extend transformations shown above with the same queries.

Next, a sequence of BAV steps analogous to ❽–❶ is generated as a sequence of contract steps, the difference from the BAV specification being that each step is accompanied by a *void* query:

❽ $\mathsf{conRel}(\langle\langle \mathsf{monitors,sno,id}\rangle\rangle, \mathit{void})$

⋮

❷ $\mathsf{conAtt}(\langle\langle \mathsf{student,name}\rangle\rangle, \mathsf{notnull}, \mathit{void})$

❶ $\mathsf{conRel}(\langle\langle \mathsf{student,id}\rangle\rangle, \mathit{void})$

3.3 Deriving GAV from BAV

In any BAV transformation which defines a global schema S_g from local schemas S_1, ..., S_n, it is possible to identify the transformation steps that define the constructs of S_g by taking the subset, $\mathcal{G}$, of the add and ext transformation steps in the total transformation sequence from $S_1 \cup \ldots \cup S_n$ to S_g.

The GAV derivation rules can then be constructed by taking each addRel/extRel step in $\mathcal{G}$, together with all addAtt/extAtt steps for the same relation, and applying the decomposition rule in reverse (*i.e.* joining the relations $\langle\langle R, a_1\rangle\rangle, \ldots, \langle\langle R, a_m\rangle\rangle$ to restore R). If the GAV rule is composed from solely addRel/addAtt BAV rules it will be exact, otherwise it will be complete.

For the example in Table 1, the steps that form $\mathcal{G}$ are ① –⑧ . Taking ① as one example of an addRel step, there are three steps ② –④ which add attributes to the student relation. This automatically gives the following definitions of the constituent parts of student:

$$
\begin{aligned}
&\langle\langle \mathsf{student,id}\rangle\rangle = \{x \mid x \in \langle\langle \mathsf{ug,id}\rangle\rangle \vee x \in \langle\langle \mathsf{phd,id}\rangle\rangle\}\\
&\langle\langle \mathsf{student,name}\rangle\rangle = \{x, y \mid\\
&\qquad \langle x, y\rangle \in \langle\langle \mathsf{ug,name}\rangle\rangle \wedge x \notin \langle\langle \mathsf{phd,id}\rangle\rangle\\
&\qquad \vee \langle x, y\rangle \in \langle\langle \mathsf{phd,name}\rangle\rangle\}\\
&\langle\langle \mathsf{student,left}\rangle\rangle = \{x, y \mid\\
&\qquad \langle x, y\rangle \in \langle\langle \mathsf{ug,left}\rangle\rangle \wedge x \notin \langle\langle \mathsf{phd,id}\rangle\rangle\\
&\qquad \vee \langle x, y\rangle \in \langle\langle \mathsf{phd,left}\rangle\rangle\}\\
&\langle\langle \mathsf{student,degree}\rangle\rangle = \{x, y \mid\\
&\qquad \langle x, y\rangle \in \langle\langle \mathsf{ug,degree}\rangle\rangle \wedge x \notin \langle\langle \mathsf{phd,id}\rangle\rangle\\
&\qquad \vee x \in \langle\langle \mathsf{phd,id}\rangle\rangle \wedge y = \text{‘phd’}\}
\end{aligned}
$$

The student relation can now be recomposed using the decomposition rule in reverse, to get a definition of the student relation in BAV constructs:

$$
\begin{aligned}
&\mathsf{student}(\mathsf{id,name,left,degree}) = \{x, y_1, y_2, y_3 \mid\\
&\qquad (x \in \langle\langle \mathsf{ug,id}\rangle\rangle \vee x \in \langle\langle \mathsf{phd,id}\rangle\rangle) \wedge\\
&\qquad (\langle x, y_1\rangle \in \langle\langle \mathsf{ug,id}\rangle\rangle \wedge x \notin \langle\langle \mathsf{phd,id}\rangle\rangle\\
&\qquad\ \vee \langle x, y_1\rangle \in \langle\langle \mathsf{phd,name}\rangle\rangle) \wedge\\
&\qquad (\langle x, y_2\rangle \in \langle\langle \mathsf{ug,left}\rangle\rangle \wedge x \notin \langle\langle \mathsf{phd,id}\rangle\rangle\\
&\qquad\ \vee \langle x, y_2\rangle \in \langle\langle \mathsf{phd,left}\rangle\rangle) \wedge\\
&\qquad (\langle x, y_3\rangle \in \langle\langle \mathsf{ug,degree}\rangle\rangle \wedge x \notin \langle\langle \mathsf{phd,id}\rangle\rangle\\
&\qquad\ \vee x \in \langle\langle \mathsf{phd,id}\rangle\rangle \wedge y_3 = \text{‘phd’})\}
\end{aligned}
$$

The ug and phd relations are then recomposed from the BAV constructs that appear on the RHS of the rule. After some logical simplification, this gives a rule that is equivalent to G_1:

$$
\begin{aligned}
&\mathsf{student}(\mathsf{id,name,left,degree}) = \{x, y_1, y_2, y_3 \mid\\
&\qquad \langle x, y_1, y_2, y_3\rangle \in \mathsf{ug} \wedge \langle x, y_1, y_2, y_3\rangle \notin \mathsf{phd}\\
&\qquad\ \vee \langle x, y_1, y_2, \_\rangle \in \mathsf{phd} \wedge y_3 = \text{‘phd’})\}
\end{aligned}
$$

By a similar process, G_2 can be derived from ⑧ and G_3 from ⑤ –⑦ .

3.4 Deriving LAV from BAV

In any BAV transformation which defines a global schema S_g from local schemas S_1, ..., S_n, it is possible to identify the transformation steps that define the constructs of some local schema S_i by taking the subset, $\mathcal{L}$, of the del and con transformations on constructs of S_i. Construction of the LAV view definitions from $\mathcal{L}$ proceeds in a similar fashion to how the GAV view definitions were constructed from $\mathcal{G}$ in the previous subsection. If the LAV rule is composed from solely delRel/delAtt BAV rules it will be exact, otherwise it will be complete.

For the example, in Table 1, the steps forming $\mathcal{L}$ for schema S_1 are ⑨ –⑮ . Rule L_1 can then be derived from ⑨ –⑩ and rule L_2 from ⑪ –⑮ .

3.5 Using BAV transformation macros

Although the BAV transformations of Table 1 form a complete definition of the integration of S_1 and S_2 into S_g, it could be argued that this has been achieved at a greater complexity compared to using GAV or LAV alone. Indeed we have shown how some transformations of the BAV pathway can derive GAV rules, and other distinct transformations derive LAV rules, and thus is could be argued that we are using the two approaches in parallel. There are two arguments against this statement. Firstly, BAV defines the integration with *more* precision than just GAV and BAV in parallel, since BAV differentiates between completely defined and partially defined constructs, rather than exact and complete rules (which in general define many constructs together). Secondly, the complexity may be greatly reduced by specifying well-known schema equivalences as **BAV transformation macros** [24]. We argue that it is possible to specify BAV transformations with as much ease as LAV or GAV derivation rules. The approach involves two phases:

1. Applying well-known schema equivalences from the schema integration literature [1] to make relations from different local databases union-compatible; this includes renaming attributes to be the same in any relations to be unioned.

2. Applying a union operation to integrate the relations into a global schema construct, possibly with a bias to prefer values from a particular local database where there is conflict between the two.

To illustrate, Table 2 shows the integration of S_1 and S_2 into S_g using this approach, where steps ㉔ – ㉗ are from phase 1 and ㉘ –㉚ are from phase 2. The schema transformations undertaken by each step are as follows:

㉔ Perform a lossless decomposition of ug in S_1 so that the sno attribute is held in a separate relation tutors. Note that the new tutors relation is union-compatible with supervises from S_2.

㉕ Extend tutor in S_1 with an attribute dept, always with value null (since there is no record of departments in S_1). This makes the tutor relation union-compatible with supervisor from S_2.

㉖ Contract the title attribute from phd, since this is not required in the global schema.

㉗ Add an attribute degree with value 'phd' to phd in S_2, making the assumption that all PhD students are registered for a PhD degree. This transformation combined with ㉔ and ㉖ makes the ug and phd relations union-compatible.

㉘ Define the student relation as a union of ug and phd, but if the same key appears in both relations keep only the attribute values that appear in phd. We term this a **right union**.

㉙ Define the staff relation as a right union of tutor and supervisor.

㉚ Define the monitors relation as a right union of tutors and supervises.

The macro used in ㉔ , which implements the decomposition of a table by moving one of its attributes Att out into a new table Rel, is defined as follows:

decompose(Rel,Att)=
  addRel(Rel, $\{x, y \mid \langle y, x\rangle \in$ Att$\}$))
  delAtt(Att, $\{x, y \mid \langle y, x\rangle \in$ NewRel$\}$)

The body of this macro forms a 'template' such that Rel can be substituted by the scheme of any relation, and Att by the scheme of any attribute. In particular, step ㉔ above expands to:

㉛ addRel($\langle\langle$tutors,sno,id$\rangle\rangle$,
    $\{x, y \mid \langle y, x\rangle \in \langle\langle$ug,sno$\rangle\rangle\}$
㉜ delAtt($\langle\langle$ug,sno$\rangle\rangle$,notnull
    $\{x, y \mid \langle y, x\rangle \in \langle\langle$tutors,sno,id$\rangle\rangle\}$)

The macro used in ㉘ –㉚ is defined as follows:

rightUnion(RU,R1,C1,R2,C2)=
  addRel(Rel,$\{\vec{x_n} \mid \langle\vec{x_n}\rangle \in$ R1 $\vee \langle\vec{x_n}\rangle \in$ R2$\}$
  for i=1 to $AttCount$(RU)
    addAtt(Att(RU, i),$\{\vec{x_n}, a_i \mid$
      $\langle\vec{x_n}, a_i\rangle \in Att$(R1, i) $\wedge \langle\vec{x_n}\rangle \notin$ R2
      $\vee \langle\vec{x_n}, a_i\rangle \in Att$(R2, i)$\}$
  conRel(R1,$\{\vec{x_n} \mid \langle\vec{x_n}\rangle \in$ RU $\wedge$ C1$\}$
  for i=1 to $AttCount$(RU)
    conAtt(Att(R1, i),$\{\vec{x_n}, a_i \mid$
      $\langle\vec{x_n}, a_i\rangle \in Att$(RU, i) $\wedge \langle\vec{x_n}\rangle \in$ R1$\}$
  delRel(R2,$\{\vec{x_n} \mid \langle\vec{x_n}\rangle \in$ RU $\wedge$ C2$\}$
  for i=1 to $AttCount$(RU)
    delAtt(Att(R2, i),$\{\vec{x_n}, a_i \mid$
      $\langle\vec{x_n}, a_i\rangle \in Att$(RU, i) $\wedge \langle\vec{x_n}\rangle \in$ R2$\}$

This forms what we term a right union relation, RU, between two union-compatible relations R1 and R2, where tuples from R1 appear only if the key values of that tuple do not appear as a tuple in R2. Defined

(24) decompose(⟨⟨ug,sno⟩⟩,⟨⟨tutors,sno,id⟩⟩)
(25) extAtt(⟨⟨tutor,dept⟩⟩,notnull, $\{x, y \mid x \in$ ⟨⟨tutor,sno⟩⟩ $\wedge\, y =$ *null*$\}$)
(26) conAtt(⟨⟨phd,title⟩⟩,notnull, *void*)
(27) addAtt(⟨⟨phd,degree⟩⟩,notnull, $\{x, y \mid x \in$ ⟨⟨phd,id⟩⟩ $\wedge\, y = \text{‘phd’}\}$)
(28) rightUnion(⟨⟨student,id⟩⟩, ⟨⟨ug,id⟩⟩, $\langle x, y\rangle \in$ ⟨⟨student,degree⟩⟩ $\wedge\, y \neq \text{‘phd’}$,
⟨⟨phd,id⟩⟩, $\langle x, y\rangle \in$ ⟨⟨student,degree⟩⟩ $\wedge\, y = \text{‘phd’}$)
(29) rightUnion(⟨⟨staff,sno⟩⟩, ⟨⟨tutor,sno⟩⟩, $\langle x, y\rangle \in$ ⟨⟨staff,dept⟩⟩ $\wedge\, y =$ *null*,
⟨⟨supervisor,sno⟩⟩, $\langle x, y\rangle \in$ ⟨⟨staff,dept⟩⟩ $\wedge\, y \neq$ *null*)
(30) rightUnion(⟨⟨monitors,sno,id⟩⟩, ⟨⟨ug,sno⟩⟩, $\langle y, x\rangle \in$ ⟨⟨monitors,sno,id⟩⟩ $\wedge\, x \notin$ ⟨⟨phd,id⟩⟩,
⟨⟨supervises,sno,id⟩⟩, $\langle x, y\rangle \in$ ⟨⟨monitors,sno,id⟩⟩ $\wedge\, y \in$ ⟨⟨phd,id⟩⟩)

Table 2. A complete BAV macro-based speci£cation of the integration of S_1 and S_2 into S_g

in the relational algebra, $\mathsf{RU} = (\mathsf{R1} - (\mathsf{R1} \ltimes \mathsf{R2})) \cup \mathsf{R2}$, where $\ltimes$ is a semi-join on the key attributes. The function $AttCount$(Rel) returns the number of non-primary key attributes of Rel. The function $Att(\mathsf{Rel}, n)$ returns the n^{th} attribute of Rel (where for the sake of argument alphabetical ordering on their names can be used to sort the attributes). The constraint C1 holds for each tuple of RU that can be determined to have originated from R1, and the constraint C2 holds for each tuple that can be determined to have originated from R2.

The transformation steps of Table 1 can, to a large extent, be generated from these macro definitions. As one might expect, the transformations are not precisely the same since the macro-based approach generates a few extra steps. None-the-less, the two integrations in Table 1 and Table 2 are logically equivalent.

In particular, (28) will expand to generate several of the transformation steps in Table 1. The addRel statement in the right-union macro will generate (1). Since the attributes of student in alphabetical order are degree, left and name, the first for loop of the macro will generate (4), (3) and (2). The conRel statement generates (15), and the second for loop generates (13), (12) and (11). The delRel statement generates (23), and the final for loop generates a step (33) not in Table 1:

(33) delAtt(⟨⟨phd,degree⟩⟩,notnull, $\{x, y \mid$
$\langle x, y\rangle \in$ ⟨⟨student,degree⟩⟩ $\wedge\, x \in$ ⟨⟨phd,id⟩⟩$\}$)

plus two steps (21) and (20) from Table 1.

Note that step (33) ‘cancels out’ step (27), in the sense that (27) introduced an attribute into the phd relation to make it union-compatible with ug, and therefore there was an extra attribute to remove from phd than was the case in Table 1. Using the rightUnion macro on (29) and (30) will produce similar ‘extra’ transformations to balance (25) and (31).

We conclude by noting that a leftUnion macro would be defined by exchanging R1 and R2 in the body of the rightUnion macro. An ‘ordinary’ union macro would be defined in a similar manner to rightUnion, except for omitting the $\langle \vec{x_n} \rangle \notin \mathsf{R2}$ constraint, and changing the delRel and delAtt to conRel and conAtt.

4 Schema Evolution

We next discuss how BAV supports the evolution of both global and local schemas, allowing the transformation pathways and schemas to be incrementally modified as opposed to having to be regenerated.

4.1 Evolution of a global schema

Suppose that n local schemas S_1, ..., S_n have been integrated into a global schema S_g. Let us consider the evolution of the global schema S_g to S'_g. The first step is to define this evolution as a schema transformation in our framework. We consider first how to handle evolutions consisting of a *single* primitive transformation.

Suppose that S_g has evolved to S'_g by a primitive transformation t. If T^{old} was the transformation from $S_1 \cup \ldots \cup S_n$ to S_g, the new transformation from $S_1 \cup \ldots \cup S_n$ to S'_g can automatically be generated by suffixing t to T^{old}:

$$T^{new} = T^{old}; t$$

There are three cases to consider for t:

1. If t is an add or del transformation, then S'_g will be semantically equivalent to S_g. Any information available from S_g is also available from S'_g, and there is nothing further that needs to be done to T^{new}.

2. If t is a contract transformation, then there will be some information that used to be present in S_g that will no longer be available from S'_g. Thus, it

may be the case that there is now no representation of some local schema construct(s) within S'_g. Nothing further needs to be done to T^{new}.

3. If t is an extend transformation with a *void* accompanying query, then the relationship of the new construct to $S_1 \dots S_n$ needs to be examined as it may actually be partially or completely derivable from $S_1 \dots S_n$ by some transformation. This requires domain knowledge e.g. from a human expert or an ontology.

   If the new construct is not derivable, then nothing further needs to be done to T^{new}. If it is derivable, then t, the last transformation step in T^{new}, needs to be replaced by a more informative extend or add step (as per our discussion in Section 2.2).

Thus, we see that the first of these two cases is handled totally automatically. The third requires domain knowledge, but is still handled as a simple evolution of the transformation pathway between $S_1 \cup \dots \cup S_n$ and S_g.

Evolutions of S_g consisting of a sequence of primitive transformation steps can be handled by applying the above treatment to each step.

4.2 Evolution of a local schema

This is a little more complex since it may require the global schema to be changed. Suppose that some S_i evolves, to S'_i say, by a primitive transformation t (again, we need only consider evolutions consisting of a single primitive transformation, and composite transformations can be handled as a series of primitive transformation steps).

If T^{old} was the transformation from $S_1 \cup \dots \cup S_i \cup \dots \cup S_n$ to S_g, a new transformation from $S_1 \cup \dots \cup S'_i \cup \dots \cup S_n$ to S_g can be automatically generated by prefixing the reverse of t to T^{old}:

$$T^{new} = \bar{t}; T^{old}$$

Again there are three cases to consider for t:

1. If t is an add or del transformation, then the new schema S'_i will be semantically equivalent to S_i. Thus any information in S_g derived from S_i can now be derived from S'_i and no changes are required to S_g or T^{new}.

2. If t is a contract transformation, then some information that used to be present in S_i will no longer be available from S'_i. In particular, it may now be the case that S_g contains constructs about which no information is derivable from any local schema — this can be determined automatically by inspection of T^{new}. Any such constructs can be removed from S_g, and the corresponding extend steps from each of the local schemas removed from T^{new}.

3. If t is an extend transformation, then the relationship of the new construct to S_g needs to be examined, as the new construct may actually be derivable from S_g by some transformation. This requires domain knowledge.

   If the new construct is indeed derivable, then the transformation that does so, T say, is appended to T^{new}, in order to add the new construct to S_g. The resulting transformation $T^{new}; T$ is equal to $\bar{t}; T^{old}; T$ which just simplifies to T^{old}.

   If the new construct is not derivable, then an extend step is appended to T^{new}, in order to add the new construct to S_g. This step matches up with the contract transformation in $\bar{t}$. This pair of steps are now removed in order for the new extent of the new construct in S'_i to be usable by queries posed on the global schema, leaving the original transformation T^{old}.

Again, the first two of the above cases can be handled totally automatically and the third semi-automatically — [20] gives further details of the second and third cases.

The above treatment also applies in the case that a new schema S_{n+1} is added to the set of local schemas from which S_g was derived, since S_{n+1} can be treated as an empty schema which is successively expanded with new constructs. Similarly, the treatment covers the removal of a local schema S_i since its constructs can be successively contracted to leave an empty schema.

5 Implementation

We are currently implementing the BAV integration approach described here within the ongoing AutoMed project at Birkbeck and Imperial Colleges (see `http://www.doc.ic.ac.uk/automed/`). Figure 2 illustrates the main components of the AutoMed system.

The Model Definitions Tool allows the modelling constructs and primitive transformations of high-level modelling languages to be specified in terms of those of the lower-level hypergraph data model (HDM). These definitions are stored in the Model Definitions Repository (MDR) [2].

The Schemas & Transformations Repository (STR) stores the schemas of data sources, intermediate schemas, global schemas, and the transformation pathways between them [2]. The Schema Transformation &

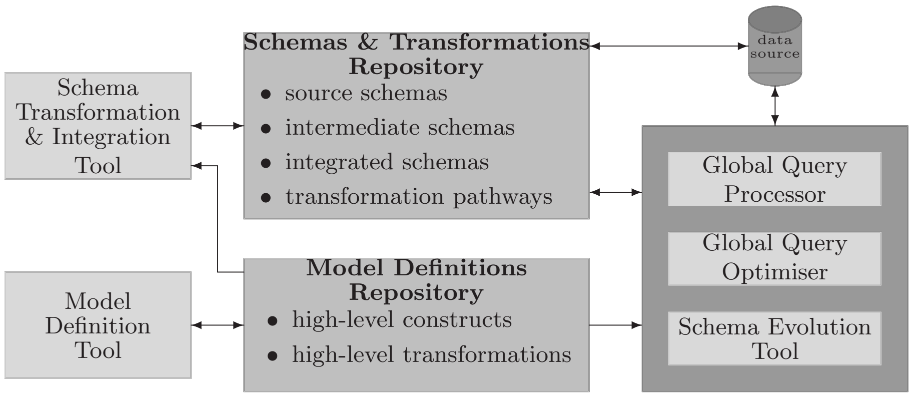


Figure 2. The AutoMed Architecture

Integration Tool allows the creation of new intermediate or global schemas together with the corresponding transformation pathways.

The Global Query Processor undertakes execution of global queries using the schemas and transformation pathways in the STR. Currently we are adopting a GAV approach to query processing [10], but from our earlier discussion in Section 3.4 a LAV approach would also be possible. We are using a functional query language as our intermediate query language (IQL) [21]. Global queries are translated into this IQL and are then reformulated to operate on the data sources, using the GAV views derived from the schema transformation pathways. After optimisation of the global query, subqueries of it are translated into the query languages supported by the data sources, and are submitted to the sources for evaluation. Any further query post-processing is undertaken by the IQL evaluator.

The Schema Evolution Tool supports the evolution of schemas and transformation pathways in the MDR, as described in Section 4 above. These evolved pathways may in some cases be amenable to simplification. For example, a composite transformation $T;t;T';\bar{t};T''$, where T,T',T'' are arbitrary sequences of primitive transformations, t is primitive transformation and $\bar{t}$ is its inverse, can be simplified to $T;T';T''$ provided that there are no references within T' to the construct being renamed, added or deleted by t. Renaming transformations may also be redundant, and hence removable. For example, $ren\ c\ c';del\ c' \equiv del\ c$, $add\ c';ren\ c'\ c \equiv add\ c$, $ren\ c'\ c'';ren\ c''\ c \equiv ren\ c'\ c$. This kind of simplification is automatically carried out by the Schema Evolution Tool [27]. More generally, these same simplifications can be applied to any schema transformation pathway generated (either automatically or manually) by the Schema Transformation & Integration Tool.

As well as implementing the above components, we are currently investigating techniques for automatic or semi-automatic generation of model definitions in the MDR and transformation pathways in the STR. For the former, we are developing a framework for defining correspondences between different high-level data models. For the latter we are investigating data mining [5] and schema matching approaches [23, 9]. We are also extending the scope of AutoMed to semi-structured data (XML, formatted files) and unstructured text. Finally, we are investigating materialised (as opposed to just virtual) data integration using AutoMed schema transformations, and in particular issues of incremental view maintenance and data lineage tracing [6].

6 Conclusions

In this paper we have presented a new approach to data integration which we term **both as view** (BAV). BAV is based on the use of reversible schema transformation sequences. We have shown that LAV and GAV view definitions can be fully derived from BAV transformation sequences, and that BAV transformation sequences can be partially derived from LAV or GAV view definitions.

BAV thus combines the benefits of GAV and LAV in the sense that any reasoning or processing which is possible with GAV or LAV view definitions will also be possible with the BAV specification.

A key advantage of BAV is that it readily supports the evolution of both local and global schemas, allowing transformation pathways and schemas to be incrementally modified as opposed to having to be regenerated.

Our original motivation for developing the BAV approach was as a general framework for schema transformation and integration, which was sufficiently expressive to capture all of the schema transformations proposed in the literature and used in practical schema integration, as well as having a formal basis. Thus, it may be argued that the transformation pathways resulting from BAV are likely to be more fine-grained, and hence more costly to reason with and process (e.g. in query optimisation and query processing) than the corresponding LAV or GAV view definitions. However, as we discussed in the previous section, BAV transformation pathways are amenable to considerable simplification. Furthermore, standard query optimisation techniques can be applied to the view definitions derived from BAV transformation pathways. Thus, we believe that query processing in AutoMed will not be significantly slower than in other LAV or GAV systems.

We are currently implementing the BAV integration approach within the ongoing AutoMed project, and this is one of the questions to be verified by practical experimentation. Our major application testbed in AutoMed is bioinformatics, and in particular the semantic integration of genomic data sources.

We are currently working on a number of other directions: development of graphical end-user tools for model definition and schema integration; techniques for automatic or semi-automatic generation of model definitions and schema transformation pathways; extension of the approach to support integration of semi-structured and unstructured information; and extension of the approach to materialised data integration.

References

[1] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[2] M. Boyd, P. McBrien, and N. Tong. The automed schema integration repository. In *Proc. BNCOD02, LNCS 2405*, pages 42–45, 2002.

[3] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, 1995.

[4] S. Chawathe *et al.* The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. 10th Meeting of the Information Processing Society of Japan*, pages 7–18, October 1994.

[5] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM SIGMOD*, 2001.

[6] H. Fan and A. Poulovassilis. Tracing data lineage using schema transformation pathways. In *Workshop on Knowledge Transformation for the Semantic Web (with ECAI'02)*, 2002.

[7] L. Fegaras and D. Maier. Towards an effective calculus for object query languages. In *ACM SIGMOD*, pages 47–58, 1995.

[8] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proc.16th National Conf. on AI*, pages 67–73. AAAI Press, 1999.

[9] M. A. Hernandez, R. J. Miller, and L. M. Haas. Clio: A semi-automatic tool for schema mapping. In *ACM SIGMOD*, 2001.

[10] E. Jasper. Global query processing in the AutoMed heterogeneous database environment. In *Proc. BNCOD02, LNCS 2405*, pages 46–49, 2002.

[11] M. Lenzerini. Data integration: A theorectical perspective. In *Proc. PODS02*, pages 247–258, 2002.

[12] A. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Logic Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.

[13] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. PODS'95*, pages 95–104. ACM Press, May 1995.

[14] A. Levy, A. Rajamaran, and J.Ordille. Querying heterogeneous information sources using source description. In *Proc. VLDB'96*, pages 252–262, 1996.

[15] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries on heterogeneous data sources. In *Proc. VLDB'01*, pages 241–250, 2001.

[16] P. McBrien and A. Poulovassilis. A formalisation of semantic schema integration. *Information Systems*, 23(5):307–334, 1998.

[17] P. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — a schema transformation approach. In *Proc. ER'99, LNCS 1728*, pages 96–113, 1999.

[18] P. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99, LNCS 1626*, pages 333–348, 1999.

[19] P. McBrien and A. Poulovassilis. A semantic approach to integrating XML and structured data sources. In *Proc. CAiSE'01, LNCS 2068*, pages 330–345, 2001.

[20] P. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Proc. CAiSE'02, LNCS 2348*, pages 484–499, 2002.

[21] A. Poulovassilis. The AutoMed Intermediate Query Language. Technical report, AutoMed Project, 2001.

[22] A. Poulovassilis and P. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.

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

[24] N. Rizopoulos. A database integration tool. Technical report, Imperial College, 2001.

[25] M. Roth and P. Schwarz. Don't scrap it, wrap it! A wrapper architecture for data sources. In *Proc. VLDB'97*, pages 266–275, Athens, Greece, 1997.

[26] M. Templeton, H.Henley, E.Maros, and D. V. Buer. InterViso: Dealing with the complexity of federated database access. *The VLDB Journal*, 4(2):287–317, 1995.

[27] N. Tong. Database schema transformation optimisation techniques for the AutoMed system. Technical report, AutoMed Project, 2002.