BAV Transformations on Relational Schemas Based on Semantic Relationships between Attributes

AutoMed Technical Report 22, Version 1

Nikolaos Rizopoulos

Monday 18th August 2003

Abstract

This report describes semantic relationships between schema elements and defines the integration of relational schemas based on the semantic relationships between their attributes using the Both-As-View integration approach. In the Both-As-View approach, data integration is based on the use of reversible schema transformation sequences that define the data mapping between schema elements. We define such sequences of transformations that could be performed on relational schemas and integrate the corresponding data sources based on the existing semantic relationships between the schemas' attributes.

1 Introduction

There have been two main approaches used in data integration methodologies to define the data mapping between the local schemas, representing the local data sources, and the global schema, defining the single integrated virtual data source. These approaches are Global-As-View (GAV) and Local-As-View (LAV). In GAV, the global schema elements are defined in terms of the local schema elements. In LAV, local elements are defined based on the global ones. One disadvantage in both of these approaches is that they don't readily support the evolution, i.e. change, of the local and global schemas. In GAV, if local schemas change then the data mapping has to be redefined, while in LAV the data mapping has to be redefined if the global schema changes.

In [2], a new integration approach is proposed, named Both-As-View (BAV), which is based on reversible sequences of schema transformation, called transformation pathways. It has been shown that using BAV it is possible to extract both GAV and LAV data mappings. In addition, BAV allows the evolution of both local and global schemas, because pathways can be incrementally modified.

In this report, we are going to use the BAV framework to define how relational databases can be integrated. Transformations are defined to be performed on the schemas based on the semantic relationships between their attributes, which are the lowest level of granularity on the relational model.

The outline of this report is as follows. We briefly talk about the existing definitions of semantic relationship between schema elements found in the literature and present our formal definitions of five types of semantic relationships in Section 2. In Section 3, we show, based on the semantic relationships between attributes, the BAV transformations that could be performed on the local, relational schemas to integrate them and produce a global relational schema. Based on the definitions in this section, Section 4 gives an example of an integration of two schemas. Finally, Section 5 gives our concluding remarks and directions of further

work.

2 Semantic Relationships

Various types of semantic relationships have been defined in the literature. Existing definitions are based on schema structure, e.g. generalization, on semantic conflicts, e.g. naming conflicts, and on real-world semantics comparison.

In [6], several types of generalization and aggregation have been defined. There are four kinds of generalization:

1. *disjoint*, when each object of the general element belongs to at most one specialized element
2. *complementary*, when each object of the general element belongs to at least one specialized element
3. *alternative*, when each object of the general element belongs to one and only specialized element
4. *general*, when there are no restrictions.

In the case of aggregation, there have been defined in [6]:

1. *simple aggregation*, which exists between elements that express properties of another element, e.g. attributes of ER entities
2. *collection aggregation*, when a collection of elements gives rise to a new element
3. *association aggregation*, when the collection of elements does not just define properties of the new element, but also their association.

In [3], semantic relationships are defined between elements by comparing their names. The generalization relationship is informally defined as the relationship between two elements A, B when A comprises B in a taxonomic sense, e.g. a generalization relationship is identified between the elements $Person, Student$. *Negative association* is identified between elements whose names are complementary, e.g. the elements *male* and *female*, incompatible or antonyms. *Positive association* is defined as the semantic relationship between elements that have synonymous names in some context or they are just frequently used in the same context.

In [7], naming conflicts are used to define semantic relationships in combination with equivalence, generalization and aggregation. Thus, two elements can be synonym or homonym (depending on the naming conflict) equivalents, generalizations or aggregations.

In [4], semantic relationship are defined based on the elements' context, their domain, i.e. the set of possible values, their extent, i.e. their current values, as well as the mapping between their domains and their extents. Five types of relationship are defined:

- *semantic equivalence* between two elements is established when there is a total 1-1 mapping between the element domains in any known and coherent context
- *semantic relationship* is established when there exists a partial many-one value mapping, or a generalization, or an aggregation abstraction [1] between the element domains

[1] One domain generalizes the other or both are generalized to a third domain. Also, one domain can be an aggregation, i.e. a collection, of the other domain.

- *semantic relevance* is defined when two elements are semantically related in one context but not in another
- *semantic resemblance* exists when the elements cannot be related but they have the same *role*, e.g. they are both entity identifiers
- *semantic incompatibility* is defined between two elements when no contexts exist associated with these elements such that the elements could have the same role.

In [5], a comparison on the set of real-world entitses that the elements represent is performed, i.e. a comparison on the real-world level and not the data-source representation. Five types of semantic relationships are specified between two relational entities/relationships A and B: A is *equal* to B, A *contains* B, A is *contained-in* B, A, B *overlap*, and A, B are *disjoint*. The relationships of [5] are identified by comparing the real-world states (RWS) of the elements, i.e. the sets of real-world entities that are represented by the elements at a given moment in time. For example, if the real-world states are equivalent then the elements are semantically equivalent. *Disjointness* is defined when the real-world states of the elements are disjoint and the elements have the same role. The equality, containment and disjointness relationships of [5] are the real-world level correspondences of semantic equivalence, semantic relationship and semantic resemblance of [4] respectively. In [8] the same kinds of relationships are identified between any two types of elements using *correspondence assertions*.

In our framework we are using the same definitions of relationships with [5], except from disjointness, and extend them with the definition of the *incompatibility* relationship. We define as $Val_{ext}(x)$ the values of an element x that are currently stored in a data source and as $Dom_{ext}(x)$ the domain of values of the element, i.e. all the possible valid values of x. We also define as $Ent_{int}(x)$ the real-world entities of x that are represented by $Val_{ext}(x)$ and as $Dom_{int}(x)$ the real-world entities represented by $Dom_{ext}(x)$. Function $Dom_{int}(x)$ is similar to RWS in [5]. Five types of semantic relationship between schema elements are identified based on the comparison of their intensional domains ($Dom_{int}(x)$). These relationships are:

1. **equivalence**: Two schema elements A and B are equivalent, $\mathsf{A} = \mathsf{B}$, iff
   $Dom_{int}(\mathsf{A}) = Dom_{int}(\mathsf{B})$
2. **subsumption**: Schema element A subsumes schema element B, $\mathsf{B} \subset \mathsf{A}$, iff
   $Dom_{int}(\mathsf{B}) \subset Dom_{int}(\mathsf{A})$
3. **overlappingness**: Two schema elements A and B are overlapping, $\mathsf{A} \simeq \mathsf{B}$, iff
   $Dom_{int}(\mathsf{A}) \cap Dom_{int}(\mathsf{B}) \neq \emptyset$
4. **disjointness**: Two schema elements A and B are disjoint, $\mathsf{A} \not\approx \mathsf{B}$, iff
   $Dom_{int}(\mathsf{A}) \cap Dom_{int}(\mathsf{B}) = \emptyset, \exists \mathsf{C} : Dom_{int}(\mathsf{A}) \cup Dom_{int}(\mathsf{B}) \subseteq Dom_{int}(\mathsf{C})$
5. **incompatibility**: Two schema elements A and B are incompatible, $\mathsf{A} \neq \mathsf{B}$, iff
   $Dom_{int}(\mathsf{A}) \cap Dom_{int}(\mathsf{B}) = \emptyset, \nexists \mathsf{C} : Dom_{int}(\mathsf{A}) \cup Dom_{int}(\mathsf{B}) \subseteq Dom_{int}(\mathsf{C})$

It is important to notice that element C in the definition of disjointness may or may not exist in the existing schemas. The notation $\exists \mathsf{C} : condition$ means that there is a real-world concept that can be represented by an existing or non-existing schema element C that satisfies the *condition*. The notation $\nexists \mathsf{C} : condition$ in the definition of incompatibility means that there is no real-world concept represented by a schema element C to satisfy the specified condition.

Throughout this report, we are going to use the term *semantically compatible* schema elements to specify that the elements are related with a semantic relationship other than incompatibility.

3 Transformations of Relational Schemas

The identification of the aforementioned semantic relationships makes the integration of two data source schemas straightforward. For each relationship there are specific transformations that can be applied. Our integration approach is similar to the four-step process described in [1]. Initially, the schemas are compared to identify the five aforementioned semantic relationships between their elements. Then, follows the *schema conforming* phase where naming conflicts are resolved by BAV rename transformations. Then schemas are superimposed by separately transforming them and producing two *union-compatible* schemas, which are defined to be identical by a sequence of BAV id transformations. One of the union-schemas is arbitrarily chosen to be the intermediate schema, which in the final *schema merging* phase is transformed based on the subsumption, overlappingness and disjointness relationships to produce the final integrated schema.

In the relational model, semantic relationships are identified between attributes. Attributes A in relation R_A of schema S_A are compared against the attributes B in relation R_B of schema S_B. As the input schemas evolve, the names of the attributes and relations change. Therefore, conditions and transformations operate on the renamed elements, which are however identified in the below definitions with their initial name. It is important to notice that in the below definitions except from the semantic relationship existing between the elements we also demand their extensional values (Val_{ext}) to display the relationship, e.g. if the elements are equivalent then their sets of extensional values must be equal as well. This is not necessary if we are certain about the semantic relationship between the elements.

The notation $\vec{x_n}$ is used as an abbreviation for the sequence of attributes $x_1, \ldots, x_n$ and the notation $\vec{x_n}$ op $\vec{y_n}$ to mean $\forall i \in \{1, \ldots, \mathsf{n}\} : x_i$ op y_i. The function $Attr(X)$ returns all the attributes $\vec{x_n}$ of relation X and function $Name(Y)$ returns the name of the element Y, which can be either an attribute or a relation. $Names(\vec{x_n})$ returns a sequence $\vec{l_n}$ of all the names of the attributes $\vec{x_n}$. The predicate unique defines that the element's name does not conflict with the name of any element in S_A or S_B and the predicate max(m) identifies the maximum m for which a relationship holds, e.g. max(m), $\vec{a_m} = \vec{b_m}$. Finally, in the rules that follow, the predicate match($\langle\langle R_X, x_1, \ldots, x_n \rangle\rangle, \langle\langle R_Y, y_1, \ldots, y_n \rangle\rangle$) defines that there is a 1-1 mapping between the aggregated instances of the attributes $x_1, \ldots, x_n$ in relation R_X and the attributes $y_1, \ldots, y_n$ in R_Y, or formally:

$$\{\langle \vec{v_n} \rangle \mid \langle \vec{k_m}, v_1 \rangle \in \langle\langle R_X, x_1 \rangle\rangle \wedge \ldots \wedge \langle \vec{k_m}, v_n \rangle \in \langle\langle R_X, x_n \rangle\rangle\}$$
$$= \{\langle \vec{v_n} \rangle \mid \langle \vec{k_m}, v_1 \rangle \in \langle\langle R_Y, y_1 \rangle\rangle \wedge \ldots \wedge \langle \vec{k_m}, v_n \rangle \in \langle\langle R_Y, y_n \rangle\rangle\}$$

where $\vec{k_m}$ represents the primary keys of the relations.

In the schema conforming phase, based on the existence and lack of equivalence relationships between attributes the following conditions must be satisfied and the corresponding transformations should be performed on the schemas:

1. lack of equivalence: $\nexists m, \vec{A_m} = \vec{B_m}$:

   condition:
   $Name(\langle\langle R_A \rangle\rangle) = Name(\langle\langle R_B \rangle\rangle)$
   $\wedge$ unique($\langle\langle R_C \rangle\rangle$)

   transformations on S_A :
   rename($\langle\langle R_A \rangle\rangle, \langle\langle R_C \rangle\rangle$)

This rule shows that relations that are not equivalent and share the same name should be renamed. The lack of any equivalence relationships between the relations' attributes determines the non-equivalence of the relations. Even if the attributes have some other type of semantic similarity relationship, the relations should still be differentiated because they remain non-equivalent and therefore should not be merged when schemas are superimposed. In this case only one relation needs to be renamed and therefore only one schema is transformed.

2. equivalence, $\mathsf{max}(m), \vec{A_m} = \vec{B_m}$:

   condition:

   $\exists \vec{a_k}$ candidate key in $R_A, \exists \vec{b_k}$ candidate key in $R_B : \vec{a_k} = \vec{b_k}$
   $\wedge\, \forall i \in \{1, \ldots, m\} : \mathsf{match}(\langle\langle R_A, a_1, \ldots, a_k, A_i \rangle\rangle, \langle\langle R_B, b_1, \ldots, b_k, B_i \rangle\rangle)$
   $\wedge\, (Name(\langle\langle R_C \rangle\rangle) = Name(\langle\langle R_B \rangle\rangle) \vee \mathsf{unique}(\langle\langle R_C \rangle\rangle))$

   transformations on S_A, S_B :

   $\forall x \in Attr(\langle\langle R_B \rangle\rangle) - \vec{B_m}, Name(x) \in Names(\vec{A_m}), \mathsf{unique}(x') :$
   $\mathsf{renameAtt}(\langle\langle R_B, x \rangle\rangle, \langle\langle R_B, x' \rangle\rangle)$
   $\forall i \in \{1, \ldots, m\} : \mathsf{renameAtt}(\langle\langle R_A, A_i \rangle\rangle, \langle\langle R_A, B_i \rangle\rangle)$
   $\mathsf{renameRel}(\langle\langle R_A \rangle\rangle, \langle\langle R_C \rangle\rangle)$
   $\mathsf{renameRel}(\langle\langle R_B \rangle\rangle, \langle\langle R_C \rangle\rangle)$

   This rule shows that two relations that are equivalent (this is determined by the equivalence of their candidate keys) should have identical names and their equivalent attributes should share the same names.

   The first condition on this rule performs a first check on the equivalence of the relations, by comparing possible candidate keys. For example, relations *person* and *place* with candidate keys $\vec{a_k}, \vec{b_k}$ respectively, might have equivalent attributes $A_1 = B_1 = \mathsf{address}$ but they are not semantically equivalent since they do not represent the same real-world concepts. The second condition on this rule performs a second test on the equivalence of the relations and the instance mapping between the candidate keys and the equivalent attributes. The attributes $\vec{a_k}, \vec{b_k}$ and $\vec{A_m}, \vec{B_m}$ might be equivalent when considered separately (a_i, b_i) but they might not be equivalent when they are aggregated. For example, two person relations with candidate keys $\vec{a_k} = \vec{b_k} = \{\mathsf{name}, \mathsf{surname}\}$ might have equivalent name and surname attributes but their mappings between these attributes' instances might differ, which makes the candidate keys not equivalent. The third condition assures that no naming conflicts will be produced by the transformations

   If the conditions are satisfied, the transformations are performed on the schemas. The first renameAtt transformations remame the attributes of R_B that have identical names with existing attributes of R_A but are not equivalent with these attributes. This assures that these attributes are not going to be merged when the schemas are superimposed. The second renameAtt transformations assign identical names to the equivalent attributes in order to merge when schemas are superimposed. The renameRel transformations show that the relations are equivalent by assigning to them identical names.

   All of these transformation have to be executed on each schema, only if the necessary elements exist. This is in order to preserve any transformations caused by other semantic relationships. For example, assume there is relation R_1 in S_A and relations R_2, R_3 in S_B with $R_1 = R_2$ and $R_1 = R_3$. The first semantic relationship would rename relations R_1 and R_2 to $R_{1,2}$. The second relationship would have to rename R_1 (which now has name $R_{1,2}$) and R_3 into $R_{1,3}$. However, if R_1, i.e. $R_{1,2}$, is renamed in schema S_A, then the same rename transformation should be performed in S_B.

3. equivalence, $\mathsf{max}(m), \vec{A_m} = \vec{B_m}$:

   condition:
   $\neg(\exists \vec{a_k}$ candidate key in $R_A, \exists \vec{b_k}$ candidate key in $R_B : \vec{a_k} = \vec{b_k}$
   $\wedge\, \forall i \in \{1, \ldots, m\} : \mathsf{match}(\langle\langle R_A, a_1, \ldots, a_k, A_i\rangle\rangle, \langle\langle R_B, b_1, \ldots, b_k, B_i\rangle\rangle)$
   $)$
   $\wedge\, Name(\langle\langle R_A\rangle\rangle) = Name(\langle\langle R_B\rangle\rangle)$
   $\wedge\, \mathsf{unique}(\langle\langle R_C\rangle\rangle)$

   transformations on S_A :
   $\mathsf{rename}(\langle\langle R_A\rangle\rangle, \langle\langle R_C\rangle\rangle)$

   This is another case where the relations are not equivalent and have to be renamed because they share the same names. The non-equivalence of the relations is determined by the fact that the candidate key and match conditions in the previous rule are not satisfied even though attributes $\vec{A_m}$ and $\vec{B_m}$ are equivalent.

After the schemas are conformed, they are superimposed, i.e. the local schemas are transformed using extend transformations, which add the *missing* elements[2] to each schema and produce an intermediate schema S_I.

The following rules define the transformations on the intermediate schema S_I:

1. subsumption, $\mathsf{max}(m), \vec{A_m} \subset \vec{B_m}$:

   condition :
   $\vec{B_m}$ candidate key in R_B
   $\wedge\, \mathsf{match}(\langle\langle R_A, A_1, \ldots, A_m\rangle\rangle, \langle\langle R_B, B_1, \ldots, B_m)\rangle\rangle$

   transformations on S_I :
   $\mathsf{addFK}(\langle\langle R_A\rangle\rangle, \langle\langle R_A, A_1\rangle\rangle, \ldots, \langle\langle R_A, A_m\rangle\rangle, \langle\langle R_B\rangle\rangle, \langle\langle R_B, B_1\rangle\rangle, \ldots, \langle\langle R_B, B_m\rangle\rangle)$

   This rule creates a foreign key constraint between the two relation because the instances of $\vec{A_m}$ can always be deduced from the candidate key $\vec{B_m}$ of R_B.

   In this case, the candidate key condition is necessary for foreign key constraints in relational models. The match condition is based on the same idea as the conditions on the second conforming rule. Here, the attributes $\vec{A_m}$ might be subsets of the attributes $\vec{B_m}$ when examined individually, so the conditions establish that the aggregation of $\vec{A_m}$ is also a subset of the aggregation of $\vec{B_m}$.

2. overlappingness, $\mathsf{max}(m), \vec{A_m} \simeq \vec{B_m}$:

   condition:
   $\exists \vec{a_k}$ candidate key in $R_A, \exists \vec{b_k}$ candidate key in R_B :
   $\vec{a_n} = \vec{b_n} \vee \vec{a_n} \subset \vec{b_n} \vee \vec{b_n} \subset \vec{a_n} \vee \vec{a_n} \simeq \vec{b_n}$
   $\wedge\, \forall i \in \{1, \ldots, m\}$:
   $\{\vec{y_k}, z \mid \langle \vec{x_n}, y_1\rangle \in \langle\langle R_A, a_1\rangle\rangle \wedge \ldots \wedge \langle \vec{x_n}, y_k\rangle \in \langle\langle R_A, a_k\rangle\rangle \wedge \langle \vec{x_n}, z\rangle \in \langle\langle R_A, A_i\rangle\rangle$
   $\wedge\, \langle \vec{x'_n}, y_1\rangle \in \langle\langle R_B, b_1\rangle\rangle \wedge \ldots \wedge \langle \vec{x'_n}, y_k\rangle \in \langle\langle R_B, a_k\rangle\rangle \wedge \langle \vec{x'_n}, z\rangle \in \langle\langle R_B, B_i\rangle\rangle\} \neq \emptyset$

[2] Missing elements are the ones that exist in one schema but not in the other

transformations on S_I :

$\mathsf{addRel}(\langle\langle C\rangle\rangle, \{\vec{y_k} \mid \langle\vec{x_n}, y_1\rangle \in \langle\langle R_A, a_1\rangle\rangle \wedge \langle\vec{x'_n}, y_1\rangle \in \langle\langle R_B, b_1\rangle\rangle$
$\wedge \ldots \wedge \langle\vec{x_n}, y_k\rangle \in \langle\langle R_A, a_k\rangle\rangle \wedge \langle\vec{x'_n}, y_k\rangle \in \langle\langle R_B, b_k\rangle\rangle$
$\wedge \langle\vec{x_n}, z_1\rangle \in \langle\langle R_A, A_1\rangle\rangle \wedge \langle\vec{x'_n}, z_1\rangle \in \langle\langle R_B, B_1\rangle\rangle$
$\wedge \ldots \wedge \langle\vec{x_n}, z_m\rangle \in \langle\langle R_A, A_m\rangle\rangle \wedge \langle\vec{x'_n}, z_m\rangle \in \langle\langle R_B, B_m\rangle\rangle\})$
$\forall i \in \{1, \ldots, k\}$:
$\mathsf{addAtt}(\langle\langle C, ab_i\rangle\rangle, \mathsf{notnull}, \{\vec{y_k}, y_i \mid \langle\vec{y_k}\rangle \in \langle\langle C\rangle\rangle\})$
$\mathsf{addPK}(\langle\langle C\rangle\rangle, \langle\langle C, ab_1\rangle\rangle, \ldots, \langle\langle C, ab_k\rangle\rangle\})$
$\forall i \in \{1, \ldots, m\}, A_i \notin \{a_1, \ldots, a_k\}$:
$\mathsf{addAtt}(\langle\langle C, c_i\rangle\rangle, \mathsf{null}, \{\vec{y_k}, z_i \mid \langle\vec{x_n}, y_1\rangle \in \langle\langle R_A, a_1\rangle\rangle \wedge \langle\vec{x'_n}, y_1\rangle \in \langle\langle R_B, b_1\rangle\rangle$
$\wedge \ldots \wedge \langle\vec{x_n}, y_k\rangle \in \langle\langle R_A, a_k\rangle\rangle \wedge \langle\vec{x'_n}, y_k\rangle \in \langle\langle R_B, b_k\rangle\rangle$
$\wedge \langle\vec{x_n}, z_1\rangle \in \langle\langle R_A, A_1\rangle\rangle \wedge \langle\vec{x'_n}, z_1\rangle \in \langle\langle R_B, B_1\rangle\rangle$
$\wedge \ldots \wedge \langle\vec{x_n}, z_m\rangle \in \langle\langle R_A, A_m\rangle\rangle \wedge \langle\vec{x'_n}, z_m\rangle \in \langle\langle R_B, B_m\rangle\rangle\})$

This rule identifies two relations that overlap and creates a new relation that represents the overlapping part.

The conditions of this rule define that the relations R_A, R_B are semantically similar since their candidate keys are similar, and actually overlap in the same manner as the overlapping attributes $\vec{A_n}$, $\vec{B_n}$. The transformations first create the relation that represents the common part of R_A, R_B, then add its primary key $\vec{ab_k}$, which is the overlapping set of R_A, R_B candidate keys, and finally add the overlapping attributes without including the candidate key attributes which have been already dealt with in the relation's primary key.

3. disjointness, $\mathsf{max}(m), \vec{A_m} \not\approx \vec{B_m}$:

condition:
$\exists \vec{a_k}$ candidate key in $R_A, \exists \vec{b_k}$ candidate key in R_B :
$\vec{a_k} \not\approx \vec{b_k}$

transformations on S_I
$\mathsf{addRel}(\langle\langle C\rangle\rangle, \{\vec{y_k} \mid (\langle\vec{x_n}, y_1\rangle \in \langle\langle R_A, a_1\rangle\rangle \wedge \ldots \wedge \langle\vec{x_n}, y_k\rangle \in \langle\langle R_A, a_k\rangle\rangle)$
$\vee (\langle\vec{x'_n}, y_1\rangle \in \langle\langle R_B, b_1\rangle\rangle \wedge \ldots \wedge \langle\vec{x'_n}, y_k\rangle \in \langle\langle R_B, b_k\rangle\rangle)\})$
$\forall i \in \{1, \ldots, k\}$:
$\mathsf{addAtt}(\langle\langle C, ab_i\rangle\rangle, \mathsf{notnull}, \{\vec{y_k}, w \mid (\langle\vec{x_n}, y_1\rangle \in \langle\langle R_A, a_1\rangle\rangle \wedge \ldots \wedge \langle\vec{x_n}, y_k\rangle \in \langle\langle R_A, a_k\rangle\rangle$
$\wedge \langle\vec{x_n}, w\rangle \in \langle\langle R_A, a_i\rangle\rangle)$
$\vee (\langle\vec{x'_n}, y_1\rangle \in \langle\langle R_B, b_1\rangle\rangle \wedge \ldots \wedge \langle\vec{x'_n}, y_k\rangle \in \langle\langle R_B, b_k\rangle\rangle$
$\wedge \langle\vec{x'_n}, w\rangle \in \langle\langle R_B, b_i\rangle\rangle)\})$
$\mathsf{addPK}(\langle\langle C\rangle\rangle, \langle\langle C, ab_1\rangle\rangle, \ldots, \langle\langle C, ab_k\rangle\rangle\})$
$\forall i \in \{1, \ldots, m\}, A_i \notin \{\vec{a_k}\}$:
$\mathsf{addAtt}(\langle\langle C, c_i\rangle\rangle, \mathsf{null}, \{\vec{y_k}, w \mid (\langle\vec{x_n}, y_1\rangle \in \langle\langle R_A, a_1\rangle\rangle \wedge \ldots \wedge \langle\vec{x_n}, y_k\rangle \in \langle\langle R_A, a_k\rangle\rangle$
$\wedge \langle\vec{x_n}, w\rangle \in \langle\langle R_A, A_i\rangle\rangle)$
$\vee (\langle\vec{x'_n}, y_1\rangle \in \langle\langle R_B, b_1\rangle\rangle \wedge \ldots \wedge \langle\vec{x'_n}, y_k\rangle \in \langle\langle R_B, b_k\rangle\rangle$
$\wedge \langle\vec{x'_n}, w\rangle \in \langle\langle R_B, B_i\rangle\rangle)\})$

In this rule, disjoint relations are unified into a new, general relation that subsumes both of the existing ones. The condition specifies that the candidate keys and therefore

S_1	ug(ug-login,ugname) phd(phd-login,pname) staff(staff-login,sname) tutors(*staff-login*,*ug-login*) supervises(*staff-login*,*phd-login*) course(course-id,cname) teaches(*staff-login*,*course-id*) registered(*ug-login*,*course-id*)

S_2	phd(phd-login) staff(staff-login,sname) course(course-id,cname) assistance(*phd-login*,*course-id*,*staff-login*,activity,date, hours)

Figure 1: Example local and global relational schemas

the relations are disjoint. The transformations create a relation which is the union of R_1, R_2 and add to it the primary keys and the disjoint attributes, similarly to the second merging rule.

In all of the above definitions, we require the maximum m ($\max(m)$) for which the relationships $\vec{A}_m$ op $\vec{B}_m$ hold and the corresponding conditions are satisfied. This is necessary in order to perform less transformations on the existing schemas.

4 Example

In order to give an example of the process, we are going to use the data sources, whose relational schemas are illustrated in Figure 1. In the figure, primary keys are underlined and foreign keys are in italics. The figure shows slightly different and cut down representations of two databases of the Computing Department at Imperial College, London. Some constraints have been defined on these schemas to make the example more illustrative of our data integration approach.

Schema S_1 shows that members of staff tutor undergraduate students and supervise PhD students. In S_1, the relation course represents all the non-laboratory courses, i.e. all the courses that have lectures in theatres. Undergraduate students register on these courses and members of staff teach them. In the college, PhD students assist in both tutorials and lab demonstrations. This is depicted in schema S_2, where course describes both courses that have tutorials in the lecture theatres and laboratory courses. Relation staff in S_2 represents the members of staff that supervise the tutorials and laboratory demonstrations and can be both lecturers or teaching associates.

We have asserted the constraints that each PhD student has to assist in at least one course and each lecturer has to teach at least one non-laboratory course. Also, non-laboratory courses might not have any tutorials nor lab demonstrations. These constraints implicitly express that the phd relations in S_1 and S_2 represent identical sets of PhD students and that staff in S_2 represents a concept that subsumes the concept of staff in S_1. In addition, the two course relations have a common set of courses, which is the courses that are teached in lecture theatres and have tutorials. Table 2 shows the existing semantic relationships between the attributes of S_1 and S_2 and the transformations that are going to be performed on the schemas based on the discussion in the previous section.

Initially, in the schema conforming phase, the naming conflicts are resolved. Schema S_1 is

S_1	vs	S_2	transformations
phd.phd-login	$=$	phd.phd-login	① – ②, ⑤
ug.ug-login	$\not\approx$	phd.phd-login	⑰ – ⑲
staff.staff-login	$\subset$	staff.staff-login	③, ⑳
staff.sname	$\subset$	staff.sname	
course.course-id	$\simeq$	course.course-id	④, ㉑ – ㉔
course.cname	$\simeq$	course.cname	

Table 1: Semantic relationships between attributes in S_1 and S_2

tranformed to S_1':

① renameAtt(⟪phd, phd-login⟫, ⟪phd, phd-login⟫)
② renameRel(⟪phd⟫, ⟪phd⟫)
③ renameRel(⟪staff⟫, ⟪S_1.staff⟫)
④ renameRel(⟪course⟫, ⟪S_1.course⟫)

and schema S_2 to S_2':

⑤ renameRel(⟪phd⟫, ⟪phd⟫)

Transformations ①, ② and ⑤ are performed because of the equivalence relationship between the key attributes phd-login in the phd relations, i.e. the second conforming rule in Section 3 . Transformations ③ and ④ are caused by the lack of equivalence relationships between the attributes of the staff and the course relations, respectively.

In the schema merging phase, S_1' and S_2' are superimposed and a intermediate union schema, S_I, is produced. Schema S_1' is going to be extended with elements that exist in S_2' but not in S_1' and schema S_2' is going to be extended similarly. The composite transformation extendTable can be used as an abbreviation in this phase:

extendTable(⟪$R, a_1, \ldots, a_n$⟫, ⟪$R, k_1, \ldots, k_m$⟫,
⟪⟪R⟫, ⟪R, fk_1⟫, ..., ⟪R, fk_l⟫,
⟪R'⟫, ⟪R', ck_1⟫, ..., ⟪R', ck_l⟫⟫⟫) =
extRel(⟪R⟫)
extAtt(⟪R, a_1⟫, null, void)
⋮
extAtt(⟪R, a_n⟫, null, void)
addPK(⟪$R, k_1, \ldots, k_m$⟫)
addFK(⟪R⟫, ⟪R, fk_1⟫, ..., ⟪R, fk_l⟫,
⟪R'⟫, ⟪R', ck_1⟫, ..., ⟪R', ck_l⟫)

Schema S_1' is extended to S_I by the following transformations:

⑥ extendTable(⟪staff,staff-login,sname⟫, ⟪staff,staff-login⟫)
⑦ extendTable(⟪course,course-id,cname⟫, ⟪course, course-id⟫)
⑧ extendTable(⟪assistance,phd-login,course-id,staff-login,activity,date,hours⟫,
⟪assistance, phd-login,course-id,staff-login,activity,date⟫,
⟪⟪assistance⟫,⟪assistance,phd-login⟫, ⟪phd⟫,⟪phd-login⟫⟫,
⟪⟪assistance⟫,⟪assistance,course-id⟫, ⟪course⟫,⟪course-id⟫⟫,
⟪⟪assistance⟫,⟪assistance,staff-login⟫, ⟪staff⟫,⟪staff-login⟫⟫)

and schema S_2' is extended to S_I:

⑨ extAtt(⟪phd,pname⟫)
⑩ extendTable(⟪ug,ug-login,ugname⟫, ⟪ug,ug-login⟫)

⑪ extendTable(⟨⟨S_1.staff,staff-login,sname⟩⟩, ⟨⟨S_1.staff,staff-login⟩⟩)
⑫ extendTable(⟨⟨tutors,staff-login,ug-login⟩⟩, ⟨⟨tutors,staff-login,ug-login⟩⟩,
⟨⟨⟨⟨tutors⟩⟩,⟨⟨tutors,staff-login⟩⟩, ⟨⟨S_1.staff⟩⟩, ⟨⟨S_1.staff, staff-login⟩⟩⟩⟩,
⟨⟨⟨⟨tutors⟩⟩,⟨⟨tutors,ug-login⟩⟩, ⟨⟨ug⟩⟩, ⟨⟨ug, ug-login⟩⟩⟩⟩)
⑬ extendTable(⟨⟨supervises,staff-login,phd-login⟩⟩, ⟨⟨supervises,staff-login,phd-login⟩⟩,
⟨⟨⟨⟨supervises⟩⟩,⟨⟨supervises,staff-login⟩⟩, ⟨⟨S_1.staff⟩⟩, ⟨⟨S_1.staff, staff-login⟩⟩⟩⟩,
⟨⟨⟨⟨phd⟩⟩,⟨⟨supervises,phd-login⟩⟩, ⟨⟨phd⟩⟩, ⟨⟨phd, phd-login⟩⟩⟩⟩)
⑭ extendTable(⟨⟨S_1.course,course-id,cname⟩⟩, ⟨⟨S_1.course,course-id⟩⟩)
⑮ extendTable(⟨⟨teaches,staff-login,course-id⟩⟩, ⟨⟨teaches,staff-login,course-id⟩⟩,
⟨⟨⟨⟨teaches⟩⟩,⟨⟨teaches,staff-login⟩⟩, ⟨⟨S_1.staff⟩⟩, ⟨⟨S_1.staff, staff-login⟩⟩⟩⟩,
⟨⟨⟨⟨teaches⟩⟩,⟨⟨teaches,course-id⟩⟩, ⟨⟨S_1.course⟩⟩, ⟨⟨S_1.course, course-id⟩⟩⟩⟩)
⑯ extendTable(⟨⟨registered,ug-login,course-id⟩⟩, ⟨⟨registered, ug-login⟩⟩,
⟨⟨⟨⟨registered⟩⟩,⟨⟨registered,ug-login⟩⟩, ⟨⟨ug⟩⟩, ⟨⟨ug, ug-login⟩⟩⟩⟩,
⟨⟨⟨⟨registered⟩⟩,⟨⟨registered,course-id⟩⟩, ⟨⟨S_1.course⟩⟩, ⟨⟨S_1.course, course-id⟩⟩⟩⟩)

Finally, in the schema restructuring phase the final integrated schema is produced. For the specific example, Table 2 shows the transformations steps from S_I to S_G and Figure 2 illustrates the two schemas, S_I and S_g.

⑰ addRel(⟨⟨student⟩⟩, $\{x \mid \langle z, x\rangle \in$ ⟨⟨ug,ug-login⟩⟩ $\vee \langle z', x\rangle \in$ ⟨⟨phd,phd-login⟩⟩$\}$)
⑱ addAtt(⟨⟨student,login⟩⟩, notnull, $\{x, w \mid (\langle z, x\rangle \in$ ⟨⟨ug,ug-login⟩⟩ $\wedge \langle z, w\rangle \in$ ⟨⟨ug,ug-login⟩⟩$)$
$\vee (\langle z', x\rangle \in$ ⟨⟨phd,phd-login⟩⟩ $\wedge \langle z', w\rangle \in$ ⟨⟨phd,phd-login⟩⟩$)\}$)
⑲ addPK(⟨⟨student⟩⟩, ⟨⟨student,login⟩⟩)
⑳ addFK(⟨⟨S_1.staff⟩⟩, ⟨⟨S_1.staff,staff-login⟩⟩, ⟨⟨S_1.staff,sname⟩⟩,
⟨⟨staff⟩⟩, ⟨⟨staff,staff-login⟩⟩, ⟨⟨S_1.staff,sname⟩⟩)
㉑ addRel(⟨⟨non-lab-assisted-course⟩⟩, $\{x \mid$
$\langle y, x\rangle \in$ ⟨⟨S_1.course,course-id⟩⟩ $\wedge \langle z, x\rangle \in$ ⟨⟨course,course-id⟩⟩
$\wedge \langle y, w_1\rangle \in$ ⟨⟨S_1.course,course-id⟩⟩ $\wedge \langle z, w_1\rangle \in$ ⟨⟨course,course-id⟩⟩
$\wedge \langle y, w_2\rangle \in$ ⟨⟨S_1.course,cname⟩⟩ $\wedge \langle z, w_2\rangle \in$ ⟨⟨course,cname⟩⟩$\}$)
㉒ addAtt(⟨⟨non-lab-assisted-course, course-id⟩⟩, notnull, $\{x, y \mid \langle x\rangle \in$ ⟨⟨non-lab-assisted-course⟩⟩)
㉓ addPK(⟨⟨non-lab-assisted-course⟩⟩, ⟨⟨non-lab-assisted-course,course-id⟩⟩)
㉔ addAtt(⟨⟨non-lab-assisted-course, cname⟩⟩, null, $\{x, w_2$
$\langle y, x\rangle \in$ ⟨⟨S_1.course,course-id⟩⟩ $\wedge \langle z, x\rangle \in$ ⟨⟨course,course-id⟩⟩
$\wedge \langle y, w_1\rangle \in$ ⟨⟨S_1.course,course-id⟩⟩ $\wedge \langle z, w_1\rangle \in$ ⟨⟨course,course-id⟩⟩
$\wedge \langle y, w_2\rangle \in$ ⟨⟨S_1.course,cname⟩⟩ $\wedge \langle z, w_2\rangle \in$ ⟨⟨course,cname⟩⟩$\}$)
$\wedge \langle x, y\rangle \in$ ⟨⟨S_1.course,cname⟩⟩ $\wedge \langle x, y\rangle \in$ ⟨⟨course,cname⟩⟩$\}$)

Table 2: Transformation pathway $S_I \rightarrow S_G$

5 Concluding Remarks

In this report, we have defined semantic relationships between schema elements and we have shown how BAV transformation pathways can be automatically derived to integrate two data sources and their schemas, when the semantic relationships between their elements are known. We have focused on the relational model and semantic relationships between attributes.

In the future, we are going to examine other modelling languages and define the transformations that should be performed on schemas of these languages. It would be very useful to examine the HDM model and define the transformations that should be performed on HDM

Figure 2: Intermediate and global relational schemas

schemas based on semantic relationships between nodes and edges. High-level models, like the relational model, and the primitive transformations on these models can be defined in terms of the HDM. Therefore, a relationship $\vec{A_m}\mathsf{op}_{rel}\vec{B_m}$ between attributes in the relational model maps to a relationship $\vec{A_m}\mathsf{op}_{HDM}\vec{B_m}$ between elements in the HDM. If the first relationship defines a transformation pathway T_{rel} and the second relationship defines T_{HDM}, then the transformation pathways should agree, i.e. the schemas produced by the transformations should be equivalent. Thus, using the definitions of BAV transformations on HDM schemas, we can examine the correctness of the transformations defined on other modelling languages for each type of semantic relationship.

References

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

[2] P. McBrien E. Jasper, N. Tong and A.Poulovassilis. View generation and optimization in AutoMed data integration framework. Technical report, AutoMed Project, 2003.

[3] P. Fankhauser, M. Kracker, and E. J. Neuhold. Semantics vs Structural Resemblance of Classes. *SIGMOD Record*, 20(4):59–63, 1991.

[4] V. Kashyap and A. Sheth. Semantic and schematic similarities between database objects: a context-based approach. *VLDB Journal*, 5(4):276–304, 1996.

[5] J.A. Larson, S.B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, April 1989.

[6] M. Castellanos M. Garcia-Solaco and F. Saltor. Discovering Interdatabase Resemblance of Classes for Interoperable Databases. In *Proceedings of the RIDE-IMS*, 1993.

[7] A. M. Ouksel and C. F. Naiman. Coordinating context building in heterogeneous information systems. *Journal of Intelligent Information Systems*, 3(2):151–183, 1994.

[8] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogenous schemas. *The VLDB Journal*, 1(1):81–126, 1992.