From Proc. CAiSE05
LNCS 3520, Pages 326-341

A General Approach to the Generation of Conceptual Model Transformations

Nikolaos Rizopoulos and Peter McBrien

Dept. Computing, Imperial College London, London SW7 2AZ
email: nr600@doc.ic.ac.uk, pjm@doc.ic.ac.uk
http://www.doc.ic.ac.uk/automed

Abstract. In data integration, a Merge operator takes as input a pair of schemas in some conceptual modelling language, together with a set of correspondences between their constructs, and produces as an output a single integrated schema. In this paper we present a new approach to implementing the Merge operator that improves upon previous work by considering a wider range of correspondences between schema constructs and defining a generic and formal framework for the generation of schema transformations. This is used as a basis for deriving transformations over high level models. The approach is demonstrated in this paper to generate transformations for ER and relational models.

1 Introduction

Initial research into data integration [1, 13] was concerned with the type of transformations that can be performed on the data source schemas [12, 24], while more recent research has focused on schema matching [15, 14, 8], *i.e.* identifying correspondences and semantic relationships between schema constructs. The process of **model management** incorporates the above by providing operators such as Match, Merge, *etc* for schemas [2]. In this paper, we are not concerned with the Match operator, which produces a set of correspondences between the schema constructs, but focus on the Merge operator, that takes as input two schemas, together with the result of Match, and produces as output a single integrated schema.

In [16, 6], schemas in a high level conceptual modelling language (such as ER, Relational, ORM, *etc*) are modelled in a nested **hypergraph data model** (**HDM**) [25, 22, 23]. We base our approach to implementing the Merge operator on determining how semantic relationships between nodes and edges in the HDM will cause transformations on the HDM to be generated, which can be mapped to BAV transformations [23, 18] on the high level modelling language. Based on these foundations, we provide a generic framework that can be used for merging schemas irrespective of the high-level modelling language used to represent them. This works by using the semantics of the high level modelling language to determine which of the low level rules may be applied.

Our methodology has the advantage of providing a generic solution to the problem of generating transformations, since it relies on the underlying graphical properties of data modelling languages, and not on the specific modelling language that is being used in a particular **universe of discourse** (**UoD**). In addition, it deals with with a variety of semantic relationships — subsumption, disjointness, intersection, and equivalence —

between schema constructs, while most existing merging techniques deal with just the equivalence semantic relationship [3, 19]. As a result, our approach does not only merge schemas but it also improves them to remove any structural redundancy.

The structure of this paper is as follows. Section 2 describes the types of semantic relationships we use as input to our Merge operator. Section 3 gives an example of how a systems integrator might use a given set of semantic relationships to perform *manually* data integration with BAV transformations. An informal justification of how the BAV transformations are derived from the semantic relationships is given, and this acts as a motivation for the generic rules. Section 4 reviews details of the HDM, and illustrates how it is used to represent the ER and relational schemas we use in this paper. Then Section 5 shows how a set of generic rules operating over the HDM may be used to generate transformations in the higher level modelling languages from semantic relationships, and in particular the transformations of Section 3. Related work is in Section 6 and our summary and conclusions are found in Section 7.

2 Semantic Relationships

Various types of semantic relationships between schema constructs have been defined in the literature. We adopt similar relationship definitions to [12], except for **disjointness**. The four types of semantic relationship between schema constructs A, B are based on the comparison of their intensional domains $D_i(A), D_i(B)$, *i.e.* the set of real world entities associated with the constructs. The relationships are:

1. **equivalence**: Two schema constructs A and B are equivalent, $\mathsf{A} \stackrel{s}{=} \mathsf{B}$, iff $D_i(\mathsf{A}) = D_i(\mathsf{B})$
2. **subsumption**: Schema construct A subsumes schema construct B, $\mathsf{B} \stackrel{s}{\subset} \mathsf{A}$, iff $D_i(\mathsf{B}) \subset D_i(\mathsf{A})$
3. **intersection**: Two schema constructs A and B are intersecting, $\mathsf{A} \stackrel{s}{\cap} \mathsf{B}$, iff $D_i(\mathsf{A}) \cap D_i(\mathsf{B}) \neq \emptyset, \exists \mathsf{C} : D_i(\mathsf{A}) \cap D_i(\mathsf{B}) = D_i(\mathsf{C})$
4. **disjointness**: Two schema constructs A and B are disjoint, $\mathsf{A} \stackrel{s}{\not\cap} \mathsf{B}$, iff $D_i(\mathsf{A}) \cap D_i(\mathsf{B}) = \emptyset, \exists \mathsf{C} : D_i(\mathsf{A}) \cup D_i(\mathsf{B}) \subseteq D_i(\mathsf{C})$

It is important to notice that construct C in the definition of intersection and disjointness may or may not exist in the schemas. The notation $\exists \mathsf{C} : condition$ means that there is a real-world concept in the domain of the data source examined, that can be represented by an existing or non-existing schema construct C that satisfies the $condition$.

3 Motivating Examples of Integration

We now present two examples of data integration, where the examples differ *only* in the modelling language being used, and not in the UoD being considered. The examples will illustrate the intuition of how schema matching performed between data sources drives the integration process and leads to integration rules. Of course the integration rules necessarily differ in detail according to the data modelling language being used, but they are triggered by the same conditions, they have common objectives and they perform analogous schema transformations.

3.1 ER Model Integration

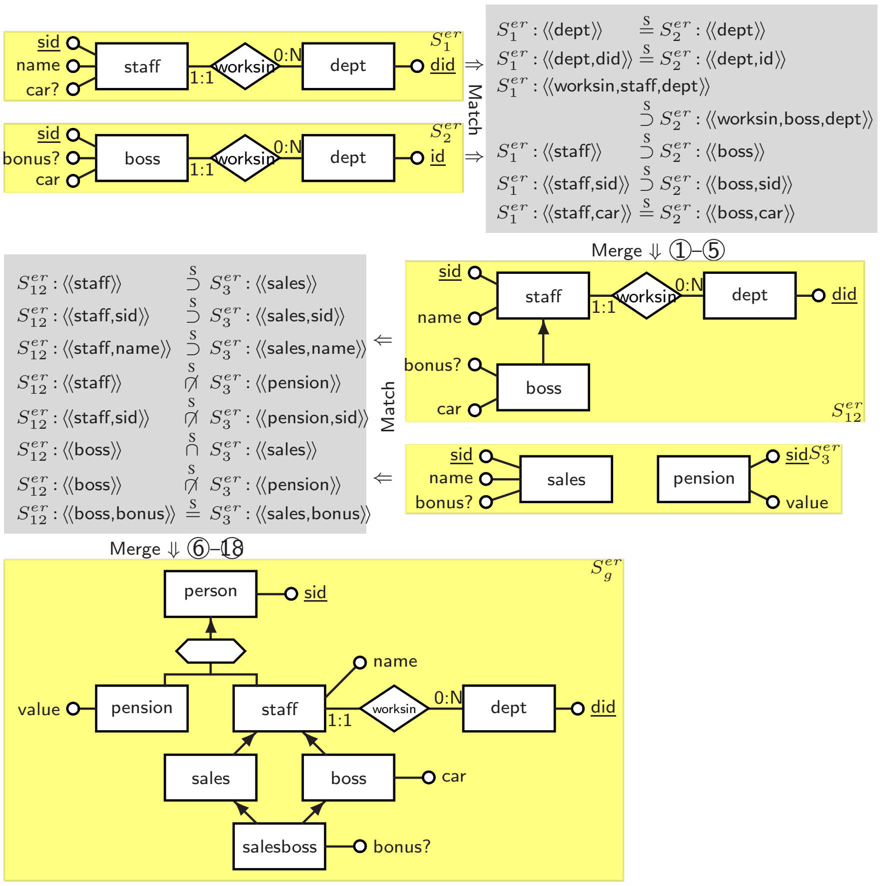


Fig. 1. Three ER models being integrated. The ER model has key attributes underlined, and optional attributes followed by a question mark. Generalisations are indicated by hexagons, and dictate that their sub-entity classes are disjoint.

Figure 1 illustrates a process where three ER schemas are integrated. First S_1^{er} and S_2^{er} are compared, and a set of semantic relationships is produced by Match. These relationships input into Merge, which integrates S_1^{er} and S_2^{er} into schema S_{12}^{er}. We then match S_{12}^{er} with S_3^{er}, producing another set of semantic relationships, which are then used to form the final global schema S_g^{er}. We will use BAV to specify the transformations necessary during data integration [18, 4], and adopt the three step conform, merge, restructure approach to schema integration [1]. Starting with integrating S_1^{er} and S_2^{er}, during the conform phase, the fact that ⟨⟨dept,did⟩⟩ attribute in S_1^{er} is equivalent to the ⟨⟨dept,id⟩⟩ attribute in S_2^{er} causes one to be renamed as the other:

1 renameAttribute(⟨⟨dept,id⟩⟩, ⟨⟨dept,did⟩⟩)

During the merging phase, the fact that the concept of ⟨⟨boss⟩⟩ in S_2^{er} is subsumed by ⟨⟨staff⟩⟩ in S_1^{er} causes a subset relation to be introduced between the two entities:

2 addSubset(⟨⟨staff,boss⟩⟩)

During the restructuring phase, we remove any redundancy that exists between the schemas. Since ⟨⟨worksin,boss,dept⟩⟩ in S_2^{er} is subsumed by ⟨⟨worksin,staff,dept⟩⟩ in S_1^{er}, we can delete the former construct without losing information in transformation 3. The fact that we are not losing information is verified by supplying a query that restores the extent of the construct we are deleting. Here we use a list comprehension [7] based language called IQL [11] used in the AutoMed system [5]. The expression in 3 states the we take those $\{\mathsf{x}\}$ values in entity ⟨⟨boss⟩⟩, and then take those $\{\mathsf{x}, \mathsf{y}\}$ found in relationship ⟨⟨worksin,staff,dept⟩⟩ with the same x value, and hence find those values of the worksin relationship that are associated to the boss entity. Also note that the rough semantics of each transformation is that the extent of the scheme in the first argument can be derived from the query that is second argument. If the first argument is a constraint, then it has no extent, and hence there is no second argument. Similarly, since ⟨⟨boss,sid⟩⟩ is subsumed by ⟨⟨staff,sid⟩⟩, we eliminate ⟨⟨boss,sid⟩⟩ in transformation 4. The fact that ⟨⟨staff,car⟩⟩ and ⟨⟨boss,car⟩⟩ are equivalent means that we should eliminate ⟨⟨staff,car⟩⟩ in 5 since it is the less specific case of the car attribute, and can state as the IQL query that its values were all those instances of ⟨⟨boss,car⟩⟩.

3 deleteRelationship(⟨⟨worksin,boss,dept,1:1,0:N⟩⟩,
    $[\{\mathsf{x}, \mathsf{y}\} \mid \{\mathsf{x}\} \leftarrow$ ⟨⟨boss⟩⟩; $\{\mathsf{x}, \mathsf{y}\} \leftarrow$ ⟨⟨worksin,staff,dept⟩⟩])

4 deleteAttribute(⟨⟨boss,sid,key⟩⟩, $[\{\mathsf{x}, \mathsf{y}\} \mid \{\mathsf{x}\} \leftarrow$ ⟨⟨boss⟩⟩; $\{\mathsf{x}, \mathsf{y}\} \leftarrow$ ⟨⟨staff,sid⟩⟩])

5 deleteAttribute(⟨⟨staff,car,null⟩⟩, ⟨⟨boss,car⟩⟩)

The resulting S_{12}^{er} is an integration of S_1^{er} and S_2^{er} that obeys one important feature of the integration rules of the framework: that **pathway** $S \rightarrow S'$ of transformations from schema S to S' satisfy the **relationship preservation property** (**RPP**). The RPP states that if the reverse pathway $P' = S' \rightarrow S$ is followed, then the relationships initially existing in S are still true, *i.e.* the semantic relationships between the constructs are preserved. Implicitly, this means that the intentional domains of the constructs are not affected by the rules, *i.e.* they do not cause any real-world entity loss nor gain. The RPP is ensured by the fact that all add and delete transformations in the pathway 1–5 that add or delete constructs that have an associated **extent** (*i.e.* set of values) are supplied with queries that fully define that extent in terms of other constructs in the schema.

Integration now proceeds to match S_{12}^{er} with S_3^{er}. Since no naming conflicts are found, we proceed directly to merging phase. The fact that there is a intersection relationship between ⟨⟨sales⟩⟩ of S_3^{er} and ⟨⟨staff⟩⟩ of S_{12}^{er} means we can introduce a common subset entity ⟨⟨salesboss⟩⟩ by transformations 7–9. The IQL expression in 7 ensures that the new entity has instances that appear in both ⟨⟨sales⟩⟩ and ⟨⟨staff⟩⟩, and this is also explicitly stated in the schema structure by the two subset constructs added by 8 and 9. The fact that there is a disjointness relationship between ⟨⟨pension⟩⟩ and ⟨⟨staff⟩⟩ means we can introduce a generalisation of them in the form of the ⟨⟨person⟩⟩ entity with transformations 10 and 11. In 10 the IQL append operator $++$ is used to append all values of ⟨⟨pension⟩⟩ to those of ⟨⟨staff⟩⟩.

6 addSubset(⟨⟨staff,sales⟩⟩)

7 addEntity(⟨⟨salesboss⟩⟩, $[\{\mathsf{x}\} \mid \{\mathsf{x}\} \leftarrow$ ⟨⟨sales⟩⟩; $\{\mathsf{x}\} \leftarrow$ ⟨⟨boss⟩⟩])

⑧ addSubset(⟨⟨sales,salesboss⟩⟩)
⑨ addSubset(⟨⟨boss,salesboss⟩⟩)
⑩ addEntity(⟨⟨person⟩⟩, ⟨⟨pension⟩⟩ ++ ⟨⟨staff⟩⟩)
⑪ addGeneralisation(⟨⟨person,pension,staff⟩⟩)

During restructuring, transformations ⑫–⑭ perform attribute specialisation, combining the equivalent ⟨⟨sales,bonus⟩⟩ and ⟨⟨boss,bonus⟩⟩ into ⟨⟨salesboss,bonus⟩⟩. Then ⑮–⑰ perform attribute generalisation, combining ⟨⟨pension,sid⟩⟩ and ⟨⟨staff,sid⟩⟩. Finally ⑱ removes the redundant ⟨⟨sales,name⟩⟩ that is subsumed by ⟨⟨staff,name⟩⟩. The result of these transformations is the final integrated schema S_g^{er} in Figure 1.

⑫ addAttribute(⟨⟨salesboss,bonus,null⟩⟩, ⟨⟨sales,bonus⟩⟩)
⑬ deleteAttribute(⟨⟨sales,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)
⑭ deleteAttribute(⟨⟨boss,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)
⑮ addAttribute(⟨⟨person,sid,key⟩⟩, ⟨⟨pension,sid⟩⟩ ++ ⟨⟨staff,sid⟩⟩)
⑯ deleteAttribute(⟨⟨pension,sid,key⟩⟩, [{x, y} | {x, y} ← ⟨⟨person,sid⟩⟩; {x} ← ⟨⟨pension⟩⟩])
⑰ deleteAttribute(⟨⟨staff,sid,key⟩⟩, [{x, y} | {x, y} ← ⟨⟨person,sid⟩⟩; {x} ← ⟨⟨staff⟩⟩])
⑱ deleteAttribute(⟨⟨sales,name,notnull⟩⟩, [{x, y} | {x, y} ← ⟨⟨staff,name⟩⟩; {x} ← ⟨⟨sales⟩⟩])

3.2 Relational Model Integration

Figure 2 shows the integration of three relational schemas S_1^{rel}, S_2^{rel} and S_3^{rel} that are equivalent to the schemas S_1^{er}, S_2^{er} and S_3^{er}. However, due to differences in the modelling language, the semantic relationships that are identified, and the integration transformations required are different from those discussed for the ER integration above. The final integrated schema S_g^{rel} is almost equivalent to the final S_g^{er} in Figure 1; the difference in semantics being that the relational model is unable to express the disjointness of ⟨⟨staff⟩⟩ and ⟨⟨pension⟩⟩ that is represented by the ER generalisation hierarchy. Our discussion of the relational integration focuses on comparing it with the ER integration, and stating why it is different. First in integrating S_1^{rel} and S_2^{rel}, the conforming phase has a transformation analogous to ①:

⑲ renameColumn(⟨⟨dept,id⟩⟩, ⟨⟨dept,did⟩⟩)

When merging S_1^{rel} and S_2^{rel} we have a transformation analogous to ②, except foreign keys are specified on the column of a table rather than the entity class (which ER subsets are defined over):

⑳ addFK(⟨⟨⟨⟨boss,sid⟩⟩,⟨⟨staff,sid⟩⟩⟩⟩)

Then during restructuring, transformation ㉑ is analogous to removing the relationship worksin in ③, but since foreign keys are constraints, no IQL query needs to be supplied. The next two steps are analogous to ④ and ⑤.

㉑ deleteFK(⟨⟨⟨⟨staff,did⟩⟩,⟨⟨dept,did⟩⟩⟩⟩)
㉒ deleteColumn(⟨⟨boss,id,notnull⟩⟩, [{x, y} | {x} ← ⟨⟨boss⟩⟩; {x, y} ← ⟨⟨staff,did⟩⟩])
㉓ deleteColumn(⟨⟨staff,car,null⟩⟩, ⟨⟨boss,car⟩⟩)

The resulting relational schema S_{12}^{rel} is equivalent to the ER schema S_{12}^{er}. Proceeding to integrate S_{12}^{rel} with S_3^{rel}, during merge, the first step is analogous to ⑥. Transformations ㉕–㉘ are similar to ⑦–⑨, but we need to additionally add a ⟨⟨sales,sid⟩⟩ column, since in the relational model each table must have key columns (whereas in the ER model, ⟨⟨salesboss⟩⟩ may inherit the sid attribute from ⟨⟨sales⟩⟩ and ⟨⟨boss⟩⟩). By a similar argument ㉙–㉜ have an extra step compared to ⑩–⑪. Also, they do not express the semantic constraint that ⟨⟨pension⟩⟩ and ⟨⟨staff⟩⟩ are disjoint.

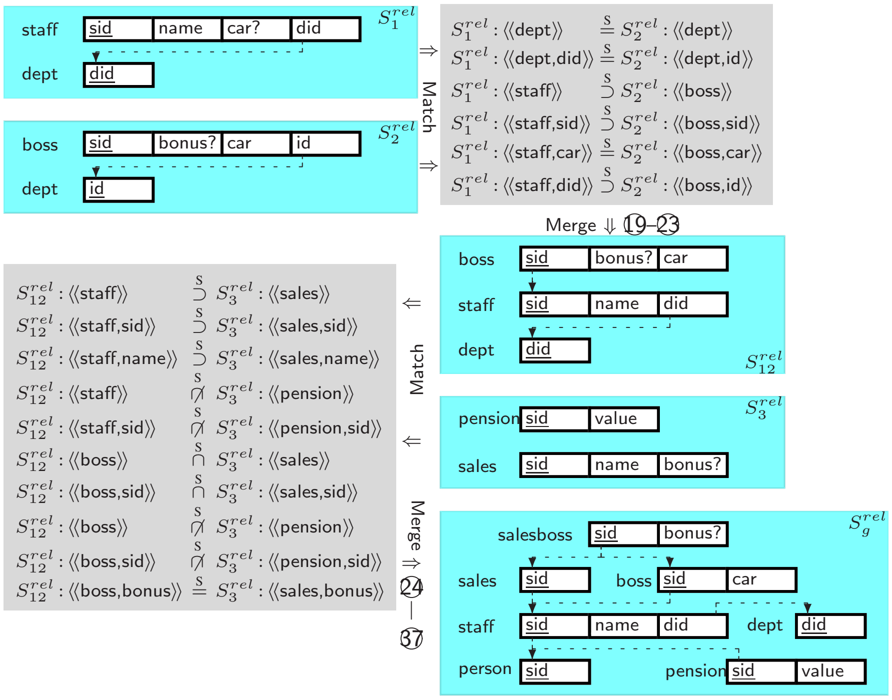


Fig. 2. Three relational models being integrated. The relational models show the columns of a table in white boxes, with the table name placed to the left of the boxes. Primary key columns are underlined, and nullable columns are followed by a question mark. Foreign keys are shown by drawing dashed arrowed lines.

24 addFK(⟨⟨⟨⟨sales,sid⟩⟩,⟨⟨staff,sid⟩⟩⟩⟩)
25 addTable(⟨⟨salesboss⟩⟩, [{x} | {x} ← ⟨⟨sales⟩⟩; {x} ← ⟨⟨boss⟩⟩])
26 addColumn(⟨⟨salesboss,sid,key⟩⟩, [{x, y} | {x, y} ← ⟨⟨sales,sid⟩⟩; {x, y} ← ⟨⟨boss,sid⟩⟩])
27 addFK(⟨⟨⟨⟨salesboss,sid⟩⟩,⟨⟨sales,sid⟩⟩⟩⟩)
28 addFK(⟨⟨⟨⟨salesboss,sid⟩⟩,⟨⟨boss,sid⟩⟩⟩⟩)
29 addTable(⟨⟨person⟩⟩, ⟨⟨pension⟩⟩ ++ ⟨⟨staff⟩⟩)
30 addColumn(⟨⟨person,sid,key⟩⟩, ⟨⟨pension,sid⟩⟩ ++ ⟨⟨staff,sid⟩⟩)
31 addFK(⟨⟨⟨⟨pension,sid⟩⟩,⟨⟨person,sid⟩⟩⟩⟩)
32 addFK(⟨⟨⟨⟨staff,sid⟩⟩,⟨⟨person,sid⟩⟩⟩⟩)

During restructuring, transformations 33–35 are analogous to 12–14. However, when combining ⟨⟨pension,sid⟩⟩ and ⟨⟨staff,sid⟩⟩ only transformation 36 can be performed — the relational analogy of 15–17 — because the columns ⟨⟨pension,sid⟩⟩ and ⟨⟨staff,sid⟩⟩ are keys and cannot be removed without making the tables invalid. Transformation 37 is analogous to 18. The result of these transformations is S_g^{rel} in Figure 2.

33 addColumn(⟨⟨salesboss,bonus,null⟩⟩, ⟨⟨sales,bonus⟩⟩)
34 deleteColumn(⟨⟨sales,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)
35 deleteColumn(⟨⟨boss,bonus,null⟩⟩, ⟨⟨salesboss,bonus⟩⟩)

㊱ addColumn(⟨⟨person,sid,key⟩⟩, ⟨⟨pension,sid⟩⟩ ++ ⟨⟨staff,sid⟩⟩)
㊲ deleteColumn(⟨⟨sales,name,notnull⟩⟩, [{x, y} |
{x, y} ← ⟨⟨staff,name⟩⟩; {x} ← ⟨⟨sales⟩⟩])

4 Representing Models in the HDM

The integration examples in the previous section show that the manual schema transformation and integration processes are driven by intuitive rules based on semantic relationships between schema constructs. The analogy of the rules for the different modelling languages imply that there are also generic rules that hold. In order to capture and define these generic rules a generic framework is necessary, *e.g.* the **hypergraph data model** (**HDM**) [23].

A **hypergraph data model** (**HDM**) M is a tuple $\langle \mathit{Nodes}, \mathit{Edges}, \mathit{Cons} \rangle$, where *Nodes* is a set of nodes of a graph, *Edges* is a set of nested hyperedges, and *Cons* is a set of constraint expressions over the *Nodes* and *Edges*. In [6] a set of primitive constraint constructs was proposed for the HDM, which will be used here in modelling a higher level modelling language in the HDM:

- **inclusion** $N_1 \subseteq N_2$: The extent of node N_1 is a subset of the extent of N_2.
- **exclusion** $\not\cap(N_1 \ldots N_n)$: For every x, y for which $1 \leq x < y \leq n$, the extent of node N_x does not intersect with the extent of N_y.
- **mandatory** $N \rhd E$: node N is connected by edge E, and every instance in the extent of N must appear at least once in the extent of E.
- **unique** $N \lhd E$: node N is connected by edge E, and every instance in the extent of N may appear no more than once in the extent of E.
- **reflexive** $N \stackrel{\text{id}}{\rightarrow} E$: when a instance of N appears in edge E, then one of the instances of E is that value of N as the value of all its nodes. Whilst by itself not very useful, reflexive combined with mandatory and unique defines a notion of a key value.

The HDM model can represent any structured data modelling language [16, 6]. Here we use the approach in [16] that classifies constructs of higher level data modelling language into one of four basic representations in the HDM, which are listed below. Table 1 shows how an illustrative subset of the constructs in Figures 1 and 2 are represented in the HDM.

A **nodal** construct is one that may appear in isolation in a model, and which has an associated extent. For example, an ER model **entity** can be created without being associated to other entities, and represents some set of objects in the UoD. Thus the entity ⟨⟨staff⟩⟩ is represented in HDM as a single node ⟨⟨staff⟩⟩. Since entities have no associated constraints, there are no constraints in the HDM. Relational **tables** are also nodal constructs, are have a very similar mapping to the HDM.

A **link** construct is one that associates other constructs with each other, and which has an extent which is drawn from those constructs. For example, the ER **relationship** construct associates existing entity constructs, and hence is a link construct. Thus, the ER ⟨⟨worksin,person,dept,1:1,0:N⟩⟩ relationship is represented in the HDM by the edge ⟨⟨worksin,person,dept⟩⟩, which is also associated with constraints that represent the relationship's cardinality constraints. For example the 1:1 role for ⟨⟨person⟩⟩ in the

node		
sch	construct	scheme
S_g^{er}	entity	⟨⟨person⟩⟩
S_g^{rel}	table	⟨⟨person⟩⟩
S_g^{er}	attribute	⟨⟨person:sid⟩⟩
S_g^{rel}	column	⟨⟨salesboss:bonus⟩⟩
S_g^{er}	entity	⟨⟨dept⟩⟩
S_g^{rel}	table	⟨⟨dept⟩⟩
S_g^{er}	entity	⟨⟨pension⟩⟩
S_g^{er}	entity	⟨⟨staff⟩⟩
S_g^{er}	entity	⟨⟨salesboss⟩⟩
S_g^{rel}	table	⟨⟨salesboss⟩⟩

edge		
sch	construct	scheme
S_g^{er}	attribute	⟨⟨_,person,person:sid⟩⟩
S_g^{rel}	column	⟨⟨_,salesboss,salesboss:bonus⟩⟩
S_g^{er}	relationship	⟨⟨worksin,staff,dept⟩⟩

cons				
sch	construct	scheme	op	scheme
S_g^{er}	relationship	⟨⟨person⟩⟩	$\triangleright$	⟨⟨worksin,person,dept⟩⟩
S_g^{er}	relationship	⟨⟨person⟩⟩	$\triangleleft$	⟨⟨worksin,person,dept⟩⟩
S_g^{rel}	foreign_key	⟨⟨person⟩⟩	$\subseteq$	⟨⟨person:sid⟩⟩
S_g^{er}	attribute	⟨⟨person⟩⟩	$\xrightarrow{\text{id}}$	⟨⟨_,person,person:sid⟩⟩
S_g^{er}	attribute	⟨⟨person⟩⟩	$\triangleleft$	⟨⟨_,person,person:sid⟩⟩
S_g^{er}	attribute	⟨⟨person⟩⟩	$\triangleright$	⟨⟨_,person,person:sid⟩⟩
S_g^{er}	attribute	⟨⟨person:sid⟩⟩	$\triangleright$	⟨⟨_,person,person:sid⟩⟩
S_g^{rel}	column	⟨⟨salesboss⟩⟩	$\triangleright$	⟨⟨_,salesboss,salesboss:bonus⟩⟩
S_g^{rel}	column	⟨⟨salesboss:bonus⟩⟩	$\triangleright$	⟨⟨_,salesboss,salesboss:bonus⟩⟩
S_g^{er}	subset	⟨⟨boss⟩⟩	$\subseteq$	⟨⟨staff⟩⟩
S_g^{er}	generalisation	⟨⟨pension⟩⟩	$\subseteq$	⟨⟨person⟩⟩
S_g^{er}	generalisation	⟨⟨staff⟩⟩	$\subseteq$	⟨⟨person⟩⟩
S_g^{er}	generalisation	⟨⟨pension⟩⟩	$\not\cap$	⟨⟨staff⟩⟩

Table 1. Representation of some constructs from S_g^{er} and S_g^{rel} in the HDM

relationship causes there to be a mandatory and unique constraint in the HDM between HDM nodes ⟨⟨person⟩⟩ and ⟨⟨worksin,person,dept⟩⟩. No constructs in the relational model are link constructs.

A **link-nodal** construct is one that has an associated extent, but may only exist when associated with some other construct. They are represented in the HDM by an edge associating a new node with some existing node or edge. For example, ER **attributes** are link-nodal constructs, and the ⟨⟨person, sid, key⟩⟩ ER attribute is represented by a node ⟨⟨person:sid⟩⟩, and a nameless edge ⟨⟨_,person,person:sid⟩⟩ linking that node to the node representing the entity ⟨⟨person⟩⟩. The fact that an attribute may not exist without its attached entity means that all attributes have a mandatory constraint between the attribute node and the edge (*e.g.* between ⟨⟨person:sid⟩⟩ and ⟨⟨_,person,person:sid⟩⟩). The key constraint is represented by all mandatory, unique and reflexive constraints between ⟨⟨person⟩⟩ and ⟨⟨_,person,person:sid⟩⟩. If the attribute had been null then the reflexive and mandatory constraints would be omitted, and if the attribute had been notnull then only the reflexive constraint would be omitted. Relational **columns** are also link-nodal constructs, and have a very similar mapping to the HDM as do ER attributes.

A **constraint** construct is one that has no extent associated with it, and just restricts the extent that other constructs may have. For example, the ER **subset** is a constraint construct, the subset ⟨⟨staff,boss⟩⟩ is represented by a subset constraint between HDM nodes ⟨⟨boss⟩⟩ and ⟨⟨staff⟩⟩. ER **generalisations** are also constraint construct, and are represented by a subset between each child entity and the parent entity, plus an exclusion between the child entities, as illustrated in Table 1 for generalisation ⟨⟨person,pension,staff⟩⟩. Relational **foreign keys** are also constraint constructs, and have a similar mapping to the HDM as do ER subsets.

5 Generic Framework for Transformation Generation

Based on the definitions in the previous section, we now define a generic framework for the integration of schemas irrespective of the high level conceptual modelling language used to represent them. We specify a set of **integration rules** that derive BAV transformations from the presence of semantic relationships between nodal, link, and link-nodal HDM constructs. These generic rules can then be translated into high level model specific rules, using techniques from [16, 6]. These higher level model rules are then applied to schemas, and generate BAV transformations such as those presented in Section 3. Four cases of generic rule to specific rule translation are identified:

1. **Exact Translation**: the generic rule can be translated into a model-specific rule by performing a one to one mapping between the HDM constructs and transformations and their model-specific equivalents, *e.g.* an addNode transformation in a generic rule would map into an addEntity transformation in the corresponding ER model rule, and an inclusion constraint would map onto a foreign key constraint in a relational model rule.
2. **Model Limitations**: in some cases the translation of a generic rule in a high level modelling language cannot be exact because a construct or a transformation in the generic rule does not have an equivalent construct or transformation in the high level language. Therefore, some conditions and/or actions of a generic rule might not be translatable. For example, the HDM exclusion constraint cannot be modelled in the relational model, and therefore the addition of such a constraint cannot be translated in a relational model rule.
3. **Meta-Constraint Requirements**: because some modelling languages have **meta-constraints**, extra conditions and actions might be necessary for the translation of a generic rule into a model-specific rule. For example, a meta-constraint of the relational model is the existence of a key column for every table. Therefore, a key column must be added by the relational model rules for every table that they add.
4. **Meta-Constraint Restrictions**: conditions and/or actions of a generic rule might be restricted in the translated model-specific rule, if they violate the meta-constraints of the modelling language the rule is translated into, *e.g.* the deletion of a link-nodal construct in a generic rule might be restricted by the corresponding relational model rule, if the link-nodal is a key column.

Since we adopt the standard conform-merge-restructure integration approach [1], integration rules for each stage must be defined. Examples of generic rules for each stage are presented next, together with explanations of their translation into high-level rules for the ER and the relational model, and their application on the schemas in Section 3.

5.1 Naming Conforming

In the first stage we deal with naming conflicts: **synonyms** when equivalent constructs have distinct names, and **homonyms** when non-equivalent constructs have identical names. Generic Merge and Distinction rules resolve these two conflicts. Two auxiliary predicates are required at this stage: $\texttt{identicalNames}(C_1, C_2)$ returns *true* when constructs C_1,C_2 have identical names, *false* otherwise, and $\texttt{uniqueName}(N)$ supplies a new name not used by any construct. For example, the Link-Nodal Merge rule:

$$\frac{\begin{array}{l} LN_1 \stackrel{s}{=} LN_2 \\ \neg\, \texttt{identicalNames}(LN_1, LN_2) \end{array}}{\mathsf{renameLN}_{gen}(LN_1, LN_2)}$$

deals with synonymous link-nodals. It examines the existence of the equivalence relationship between two link-nodals LN_1 and LN_2 with non-identical names and assigns to them a common name. The Link-Nodal Distinction rule:

$$\frac{\begin{array}{l} \neg\, LN_1 \stackrel{s}{=} LN_2 \\ \texttt{identicalNames}(LN_1, LN_2) \\ \texttt{uniqueName}(LN') \end{array}}{\mathsf{renameLN}_{gen}(LN_1, LN')}$$

deals with homonym link-nodals. It assigns to one of them a unique name to explicitly make the two link-nodals distinct. The Nodal and Link Merge and Distinction rules are defined in the same manner.

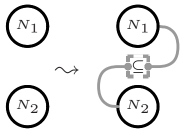
These generic naming conforming rules can be translated into high level models by Exact Translation. Simply, the generic rename transformations will be replaced by the model-specific rename transformations [16]. For example, the Attribute and Column Merge rules for the ER and the relational model are produced from the generic Link-Nodal Merge rule by replacing $\mathsf{renameLN}_{gen}$ with renameAttribute and renameColumn, respectively. In the examples of the previous sections, applying these rules would result into transformations 1 and 19, respectively.

The naming conforming rules satisfy the RPP since the intentional domain of the constructs is not affected, only equivalent constructs are assigned identical names.

5.2 Schema Merging

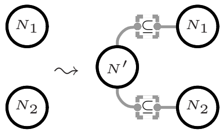
In the next stage of the integration, the schemas are merged and a single schema is produced. Pair of equivalent constructs, which now have identical names, collapse into single constructs, new constructs are added and constraints are introduced. The purpose of the rules of this stage is to identify any possible concepts that do not appear explicitly in the schemas. The rules satisfy the RPP since constructs are not deleted from the schema, only added. Therefore the intentional domain of the existing constructs is not affected.

$$\frac{\langle\langle N_2\rangle\rangle \overset{s}{\subset} \langle\langle N_1\rangle\rangle}{\mathsf{addCons}(N_2 \subseteq N_1)}$$



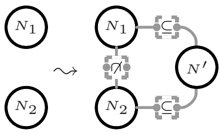
(a) Inclusion Introduction

$$\frac{\begin{array}{l}\langle\langle N_1\rangle\rangle \overset{s}{\cap} \langle\langle N_2\rangle\rangle \\ \mathtt{uniqueName}(N')\end{array}}{\begin{array}{l}\mathsf{addNodal}_{gen}(\langle\langle N'\rangle\rangle, [\{x\} \mid \{x\} \leftarrow \langle\langle N_1\rangle\rangle; \{x\} \leftarrow \langle\langle N_2\rangle\rangle]) \\ \mathsf{addCons}(\langle\langle N'\rangle\rangle \subseteq \langle\langle N_1\rangle\rangle) \\ \mathsf{addCons}(\langle\langle N'\rangle\rangle \subseteq \langle\langle N_2\rangle\rangle)\end{array}}$$



(b) Addition of Intersection

$$\frac{\begin{array}{l}\langle\langle N_1\rangle\rangle \overset{s}{\not\cap} \langle\langle N_2\rangle\rangle \\ \mathtt{uniqueName}(N')\end{array}}{\begin{array}{l}\mathsf{addNodal}_{gen}(\langle\langle N'\rangle\rangle, \langle\langle N_1\rangle\rangle \ ++ \ \langle\langle N_2\rangle\rangle) \\ \mathsf{addCons}(\langle\langle N_1\rangle\rangle \subseteq \langle\langle N'\rangle\rangle) \\ \mathsf{addCons}(\langle\langle N_2\rangle\rangle \subseteq \langle\langle N'\rangle\rangle) \\ \mathsf{addCons}(\langle\langle N_1\rangle\rangle \not\cap \langle\langle N_2\rangle\rangle)\end{array}}$$



(c) Addition of Union

Fig. 3. Generic Schema Merging Rules

The integration rules at this stage examine the existence of subsumption, intersection and disjointness relationships between nodal constructs. When a subsumption relationship is identified between two nodals then an inclusion constraint must be added between them (Figure 3(a)). When two nodals N_1,N_2 intersect, then a new nodal should be added to represent the common intentional domain of N_1 and N_2. The appropriate inclusion constraints must also be introduced as illustrated in Figure 3(b). Finally, when two nodals are disjoint, an exclusion constraint is added between them and the union nodal is introduced to represent the union of the disjoint nodal domains (Figure 3(c)).

Exact Translation can be applied on these rules to produce the ER corresponding ones. The $\mathsf{addNodal}_{gen}$ actions would translate into addEntity transformations and the addition of inclusion constraints would become addSubset transformations. In Section 3, the ER Inclusion Introduction rule generates transformation 2 and the ER Addition of Introduction generates 7–9 transformations. Finally, the three HDM constraints in the Addition of Union rule map onto an ER generalization, therefore the ER Addition of Union rule can also be produced by Exact Translation. The complete rule, which in our examples generates transformations 10–11, is defined next:

$$\frac{\begin{array}{l}\langle\langle E_1\rangle\rangle \overset{s}{\not\cap} \langle\langle E_2\rangle\rangle \\ \mathtt{uniqueName}(E')\end{array}}{\begin{array}{l}\mathsf{addEntity}(\langle\langle E'\rangle\rangle, \langle\langle E_1\rangle\rangle \ ++ \ \langle\langle E_2\rangle\rangle) \\ \mathsf{addGeneralisation}(\langle\langle E', E_1, E_2\rangle\rangle)\end{array}}$$

Producing the corresponding merging rules for the relational model does not only require Exact Translation but there is also a Meta-Constraint Requirement and a Model Limitation case. For example, if we examine the Addition of Union rule we have that the generic $\mathsf{addNodal}_{gen}$ would become an addTable transformation by Exact Translation. Because of the Meta-Constraint Requirement of the relational model that each table must have a key column, the rule is required to perform an extra addColumn transformation. Conditions `keyColumn` that identify the key columns of the disjoint tables are also additionally added. Notice that the constraints added by the generic rule cannot be represented entirely in the relational model. The Model Limitation is the exclusion constraint, which does not have a corresponding construct in the relational model. Therefore, only the addition of the inclusion constraints is translated (into addition of foreign keys). The complete rule is defined below. An application of it can be seen in transformations 29–32.

$$
\begin{array}{l}
\langle\langle T_1\rangle\rangle \overset{s}{\not\cap} \langle\langle T_2\rangle\rangle \\
\texttt{uniqueName}(T') \\
\texttt{uniqueName}(KC') \\
\texttt{keyColumn(}\langle\langle T_1\rangle\rangle\texttt{,}\langle\langle T_1, KC_1\rangle\rangle\texttt{)} \\
\texttt{keyColumn(}\langle\langle T_2\rangle\rangle\texttt{,}\langle\langle T_2, KC_2\rangle\rangle\texttt{)} \\
\hline
\mathsf{addTable}(\langle\langle T'\rangle\rangle, \langle\langle T_1\rangle\rangle\ ++\ \langle\langle T_2\rangle\rangle) \\
\mathsf{addColumn}(\langle\langle T', KC', \mathsf{key}\rangle\rangle, \langle\langle T_1, KC_1\rangle\rangle\ ++\ \langle\langle T_2, KC_2\rangle\rangle) \\
\mathsf{addFK}(\langle\langle\langle\langle T_1, KC_1\rangle\rangle, \langle\langle T', KC'\rangle\rangle\rangle\rangle) \\
\mathsf{addFK}(\langle\langle\langle\langle T_2, KC_2\rangle\rangle, \langle\langle T', KC'\rangle\rangle\rangle\rangle)
\end{array}
$$

5.3 Schema Restructuring

In the final stage of the integration, the schema produced during merging is restructured in order to remove structural redundancies. The restructuring rules are defined based on the identified semantic relationships between links and link-nodals. For each relationship between links or link-nodals, all the possible relationships between the corresponding attached nodes are examined. All the possible constraint configurations are also considered. We illustrate this approach with two examples.

$$
\begin{array}{l}
\langle\langle E_2, N_2, N'_{1/2}\rangle\rangle \overset{s}{\subset} \langle\langle E_1, N_1, N'_{1/2}\rangle\rangle \\
\langle\langle N_2\rangle\rangle \overset{s}{\subset} \langle\langle N_1\rangle\rangle \\
\langle\langle N_1\rangle\rangle \lhd \langle\langle E_1, N_1, N'_{1/2}\rangle\rangle \\
\langle\langle N_2\rangle\rangle \rhd \langle\langle E_2, N_2, N'_{1/2}\rangle\rangle \\
\texttt{constraints(\_,}\langle\langle E_2, N_2, N'_{1/2}\rangle\rangle\texttt{,}Cons\texttt{)} \\
\hline
\texttt{deleteConstraints(}Cons\texttt{)} \\
\texttt{moveDependents(}\langle\langle E_2, N_2, N'_{1/2}\rangle\rangle\texttt{,}\langle\langle E_1, N_1, N'_{1/2}\rangle\rangle\texttt{)} \\
\mathsf{deleteLink}_{gen}(\langle\langle E_2, N_2, N'_{1/2}\rangle\rangle, [\{x,y\} \mid \{x,y\} \leftarrow \langle\langle E_1, N_1, N'_{1/2}\rangle\rangle; \{x\} \leftarrow \langle\langle N_2\rangle\rangle])
\end{array}
$$

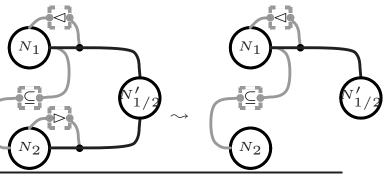


Fig. 4. Generic Optional Link Removal

Figure 4 examines one case of link subsumption and defines the Generic Optional Link Removal rule. More specifically link $e_1 = \langle\langle E_1, N_1, N'_{1/2}\rangle\rangle$ subsumes link $e_2 = \langle\langle E_2, N_2, N'_{1/2}\rangle\rangle$ and node $\langle\langle N_1\rangle\rangle$ subsumes $\langle\langle N_2\rangle\rangle$. Since the domain of e_2 is subsumed by e_1, link e_2 can be considered for deletion. In order to be able to fully restore e_2 after its deletion and hence to satisfy the RPP, it must be ensured that the entities of e_1 that do not appear in e_2 associate with $\langle\langle N'_{1/2}\rangle\rangle$ only the entities of $\langle\langle N_1\rangle\rangle$ that do not appear in $\langle\langle N_2\rangle\rangle$. If this restriction is true then e_2 can be restored by identifying the entities of e_1 that are associated with entities of $\langle\langle N_2\rangle\rangle$. The constraints that force this restriction are: $\langle\langle N_1\rangle\rangle \lhd \langle\langle E_1, N_1, N'_{1/2}\rangle\rangle$ and $\langle\langle N_2\rangle\rangle \rhd \langle\langle E_2, N_2, N'_{1/2}\rangle\rangle$. Notice that before the link is deleted any constructs that depend on it have to be examined. Dependent constraints, identified by `constraint`, are deleted and all other dependent constructs are moved to the remaining link.

The translation of this generic Optional Link Removal rule in the ER language is a simple Exact Translation:

$$\frac{\begin{array}{l}\langle\langle E_2, N_2, N'_{1/2}, \mathtt{1:}N, Card2\rangle\rangle \overset{\mathrm{s}}{\subset} \langle\langle E_1, N_1, N'_{1/2}, \mathtt{0:1}, Card1\rangle\rangle \\ \langle\langle N_2\rangle\rangle \overset{\mathrm{s}}{\subset} \langle\langle N_1\rangle\rangle \\ \mathtt{constraints(\_,}\langle\langle E_2, N_2, N'_{1/2}\rangle\rangle\mathtt{,}Cons\mathtt{)}\end{array}}{\begin{array}{l}\mathtt{deleteConstraints(}Cons\mathtt{)} \\ \mathtt{moveDependents(}\langle\langle E_2, N_2, N'_{1/2}\rangle\rangle\mathtt{,}\langle\langle E_1, N_1, N'_{1/2}\rangle\rangle\mathtt{)} \\ \mathsf{deleteRelationship}(\langle\langle E_2, N_2, N'_{1/2}, \mathtt{1:N}, Card2\rangle\rangle, [\{x, y\} \mid \\ \quad \{x, y\} \leftarrow \langle\langle E_1, N_1, N'_{1/2}\rangle\rangle; \{x\} \leftarrow \langle\langle N_2\rangle\rangle])\end{array}}$$

The HDM $\mathsf{deleteLink}_{gen}$ becomes an $\mathsf{deleteRelationship}$ transformation and the mandatory and unique constraints map to cardinality constraints as explained in [6]. The constraints between $\langle\langle N_1\rangle\rangle$ and e_1 map into a `0:1` cardinality constraint, which is less restrictive than `1:1`, and the mandatory constraint between $\langle\langle N_2\rangle\rangle$ and e_2 maps into a `1:N` cardinality constraint. In our examples, an application of the ER Optional Link Removal rule generates transformation 3.

Another example of a restructuring rule is illustrated in Figure 5. The case that is examined here is the existence of a disjointness relationship between link-nodal constructs $\langle\langle X_1, N_1\rangle\rangle$, $\langle\langle X_2, N_2\rangle\rangle$ when $\langle\langle X_1\rangle\rangle$,$\langle\langle X_2\rangle\rangle$ are also disjoint. In this case, the link-nodal constructs can be generalized by moving them from the sub-nodes $\langle\langle X_1\rangle\rangle$,$\langle\langle X_2\rangle\rangle$ to the union node $\langle\langle X'\rangle\rangle$ added during the merging stage and identified by predicate `createdNodal`. The rule adds the union link-nodal onto $\langle\langle X'\rangle\rangle$ and then deletes the existing link-nodals. Translating this rule to a high level model (such as transformations 15–17 in the ER model) requires an examination of Meta-Constraint Restrictions, except from performing Exact Translation of the BAV transformations.

For the ER model, the predicate $\mathtt{addLN}_{gen}$ can be redefined by Exact Translation. Before performing the corresponding high level transformation, *i.e.* addAttribute, the common constraints of the existing attributes must be identified and cascaded into the new attribute. Also note that the $\mathtt{deleteLN}_{er}$ must implement the meta-constraint that either the attribute is not key, or its attached entity is a child of a subset or a generalisation.

$\langle\langle X_1, N_1 \rangle\rangle \overset{s}{\not\cap} \langle\langle X_2, N_2 \rangle\rangle$
$\langle\langle X_1 \rangle\rangle \overset{s}{\not\cap} \langle\langle X_2 \rangle\rangle$
createdNodal(X_1, X_2, X')
uniqueName(N')

---

addLN$_{gen}$($\langle\langle X', N' \rangle\rangle$,$[\langle\langle X_1, N_1 \rangle\rangle] ++ [\langle\langle X_2, N_2 \rangle\rangle]$,
$\langle\langle X_1, N_1 \rangle\rangle$,$X_2, N_2$)
deleteLN$_{gen}$($\langle\langle X_1, N_1 \rangle\rangle, [\{x, y\} \mid$
$\{x, y\} \leftarrow \langle\langle X', N' \rangle\rangle; \{x\} \leftarrow \langle\langle X_1 \rangle\rangle]$)
deleteLN$_{gen}$($\langle\langle X_2, N_2 \rangle\rangle, [\{x, y\} \mid$
$\{x, y\} \leftarrow \langle\langle X', N' \rangle\rangle; \{x\} \leftarrow \langle\langle X_2 \rangle\rangle]$)

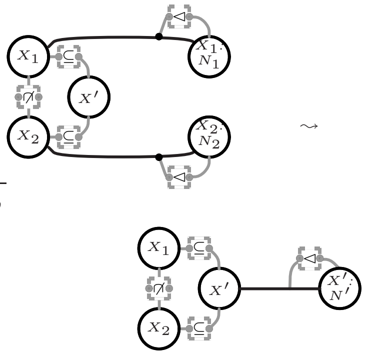


Fig. 5. Link-Nodal Generalisation

addLN$_{er}$($\langle\langle X, N \rangle\rangle$,$Q$,$\langle\langle X_1, N_1, C_1 \rangle\rangle$,$\langle\langle X_2, N_2, C_2 \rangle\rangle$) :-
    C_1=C_2, addAttribute($\langle\langle X, N, C_1 \rangle\rangle, Q$).
deleteLN$_{er}$($\langle\langle X, N, C \rangle\rangle$,$Q$) :-
    $\neg C = \mathsf{key}, \langle\langle X', X \rangle\rangle, \langle\langle X', \ldots, X, \ldots \rangle\rangle$, deleteAttribute($\langle\langle X, N, C \rangle\rangle$,$Q$).

Translating the rule in the relational modelling language, an extra restriction is required. In the redefinition of addLN$_{gen}$ a new column cannot be added if it is the union of key columns, because table $\langle\langle N' \rangle\rangle$ has already got a key column, added by the Addition of Union rule. In the case of deleteLN$_{rel}$, a Meta-Constraint Restriction applies which does not allow the deletion of key columns.

addLN$_{rel}$($\langle\langle X, N \rangle\rangle$,$Q$,$\langle\langle X_1, N_1, C_1 \rangle\rangle$,$\langle\langle X_2, N_2, C_2 \rangle\rangle$) :-
    C_1=C_2, $\neg$ C_1=key, addColumn($\langle\langle X, N, C_1 \rangle\rangle, Q$).
deleteLN$_{rel}$($\langle\langle X, N, C \rangle\rangle$,$Q$) :-
    $\neg$ $C = \mathsf{key}$, deleteColumn($\langle\langle X, N, C \rangle\rangle$,$Q$).

6 Related Work

Many approaches to generating schema transformations can be found in the literature. Early work can be found in [12, 21], where formal definitions of semantic relationships between schema constructs similar to ours are given. However, both approaches are concerned with integrating schemas defined in an extended ER language, which induces restrictions compared to our generic approach of using the low-level HDM. We define a wider set of formal rules and examine all possible constraint configurations.

In [10] similar semantic relationships to ours are used, where schema integration is performed based on corresponding ontologies and concepts. However, the steps for the creation of the integrated schema are not formally defined, nor is the data mapping, and further restrictions are imposed, *e.g.* one schema construct can only map to only one other construct.

The work most related to ours is [3], where a low-level graph-based modelling language is also adopted, called Vanilla, which models both the schemas and the correspondences between their constructs. There is an example showing how an extended ER language can be supported by Vanilla, however there is no extensive explanation of how schemas can be translated from Vanilla into a high level modelling language. The advantage of using the HDM as the common modelling language is that the translation to and from Relational, ER, XML and UML schemas [16, 17] has already been studied. Additionally, the schema integration approach in [3] is based only on semantic equivalence between nodes, while we deal with a wider range of semantic relationships between all types of generic constructs (nodals, links and link-nodals). However, the advantage of [3] is that data-level correspondences between constructs are also considered, *e.g.* data level correspondence would specify that the instances of two constructs can be concatenated. Our approach has a more semantic perspective than a data-level one. Another difference between the two approaches is that we explicitly deal with constraints and they are a necessary part of our rules. Finally, as for [19] where another schema integration approach is proposed based entirely on equivalence relationships, our methodology has the advantage of not only removing integrating schemas but additionally removing structural redundancies.

7 Summary and Conclusions

In this paper, we have presented a generic and formal framework to generate schema transformations in the Merge operator. We use the low level HDM as the common data modelling language, which permits the extension of this framework to any higher-level modelling language. Our integration rules take as input four types of semantic relationships — equality, subsumption, disjointness and intersection — and generate BAV transformations over the HDM. Using the correspondence between the HDM and higher-level models, these rules can be translated into rules that apply to higher-level models. In this paper examples of translating generic rules into ER and relational modelling language rules have been presented.

Since we adopt the BAV integration methodology, we are able to reason about the transformation steps, demonstrate that we preserve information during the integration and prove the correctness of the process.

Since we deal with a wide variety of semantic transformations, our framework can be used in conjunction with most schema matching techniques [14, 15] both for merging and improving schema structure.

In future work we will consider more complicated mappings as in [8, 20, 9], and rules for removing redundant constraints. Relationships between different types of constructs might also prove useful. Our target is to implement a tool that based on this formal framework can assist in the automatic integration of schemas.

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. Bernstein. Applying model management to classical meta data problems. In *Proc. CIDR, 2003*, 2003.
3. Philip A. Bernstein and Rachel A. Pottinger. Merging models based on given correspondences. In *Proc. 29th VLDB Conference*, Berlin, 2003.

4. M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *Proc. CAiSE2004*, volume 3084 of *LNCS*, pages 82–97. Springer-Verlag, 2004.
5. M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. Overview of the automed repository. Technical Report No. 26, AutoMed, 2004.
6. M. Boyd and P.J. McBrien. Towards a semi-automated approach to intermodel transformations. In *Proc. EMMSAD 04, CAiSE Workshop Proceedings Volume 1*, pages 175–188, 2004.
7. P. Buneman et al. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.
8. Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. iMAP: discovering complex semantic matches between database schemas. In *Proc. SIGMOD 2004*, pages 383–394. ACM Press, 2004.
9. Anca Dobre, Farshad Hakimpour, and Klaus R. Dittrich. Operators and classification for data mapping in semantic integration. In *Proc. ER 2003*, pages 534–547, 2003.
10. F. Hakimpour and A. Geppert. Global schema generation using formal ontologies. In *Proc. ER02*, volume 2503 of *LNCS*, pages 307–321. Springer-Verlag, 2002.
11. E. Jasper, N. Tong, P.J. McBrien, and A. Poulovassilis. View generation and optimisation in the AutoMed data integration framework. In *Proc. Baltic DB&IS04*, volume 672 of *Scientific Papers*, pages 13–30. Univ. Latvia, 2004.
12. 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.
13. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.
14. L.Xu and D.W. Embley. Discovering direct and indirect matches for schema elements. In *8th International Conference on Database Systems for Advanced Applications (DASFAA '03), Kyoto, Japan, March 26–28, 2003*, pages 39–46, 2003.
15. J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proc. 27th VLDB Conference*, pages 49–58, 2001.
16. P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer, 1999.
17. P.J. McBrien and A. Poulovassilis. A semantic approach to integrating XML and structured data sources. In *Proc. CAiSE'01*, volume 2068 of *LNCS*, pages 330–345. Springer, 2001.
18. P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.
19. Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. In *Proc. SIGMOD 2003*, pages 193–204. ACM Press, 2003.
20. R.J. Miller, M.A. Hernández, L.M. Haas, L.-L. Yan, C.T.H. Ho, R. Fagin, and L. Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.
21. C. Parent and S. Scappapoetra. View integration: A step forward in solving structural conflicts. Research Report, EPFL-Computer Sc. Dept. Lausanne, 1990.
22. A. Poulovassilis and M. Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Trans. on Information Systems*, 12(1):35–68, 1994.
23. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
24. S. Scappapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, April 1994.
25. C. Zaniolo and M. Melkanoff. A formal approach to the definition and the design of conceptual schemata for database systems. *ACM TODS*, 1982.