Towards a Semi-Automated Approach to Intermodel Transformation

AutoMed Technical Report No. 29, Version 2

Michael Boyd
PSA Parts Ltd
London SW19 3UA
mb@psaparts.co.uk

Peter McBrien
Imperial College London
London SW7 2AZ
pjm@doc.ic.ac.uk

3rd December 2004

Abstract

Data integration is frequently performed between heterogeneous data sources, requiring that not only the structure of the data, but also the data modelling language in which that structure is represented must be transformed between one data source and another data source.

This paper describes an extension to the hypergraph data model (HDM), used in the AutoMed data integration approach, that allows constraint constructs found in static data modelling languages to be represented by a small set of primitive constraint operators in the HDM. In addition, a set of five equivalence preserving transformation rules are defined that operate over this extended HDM and are demonstrated to allow a mapping between equivalent relational, ER, UML and ORM models to be defined.

The approach we propose provides a precise framework in which to compare data modelling languages, and precisely identifies what semantics of a particular domain one data model may express that another data model may not express. The approach also forms the platform for further work in automating the process of transforming between different data modelling languages. The use of the both-as-view approach to data integration means that a bidirectional association is produced between data models. Hence a further advantage of the approach is that closure of data mappings may be performed such that mapping two data models to one common data model will produce a bidirectional mapping between the original two data sources.

keywords: conceptual data modelling, mappings, transformations, multiple representations.

1 Introduction

The AutoMed project has developed the first implementation [2, 12] of a data integration technique called **both-as-view** (**BAV**) [16], which subsumes the expressive power of other published data integration techniques such as **global-as-view** (**GAV**), **local-as-view** (**LAV**), and **global-local-as-view** (**GLAV**) [14].

The AutoMed system also distinguishes itself in being an approach which has a clear methodology for handling a wide range of static data modelling languages in the integration process [15], as opposed to the other approaches that assume integration is always performed in a single common data model. This is achieved by allowing a user to relate the modelling constructs of a higher level modelling language such as ER, relational, UML, or ORM, to the constructs in a single lower level common data modelling language called the **hypergraph data model** (**HDM**) [21]. Figure 1 illustrates this concept being applied to four higher level models, all of which are related to the same underlying HDM graph.

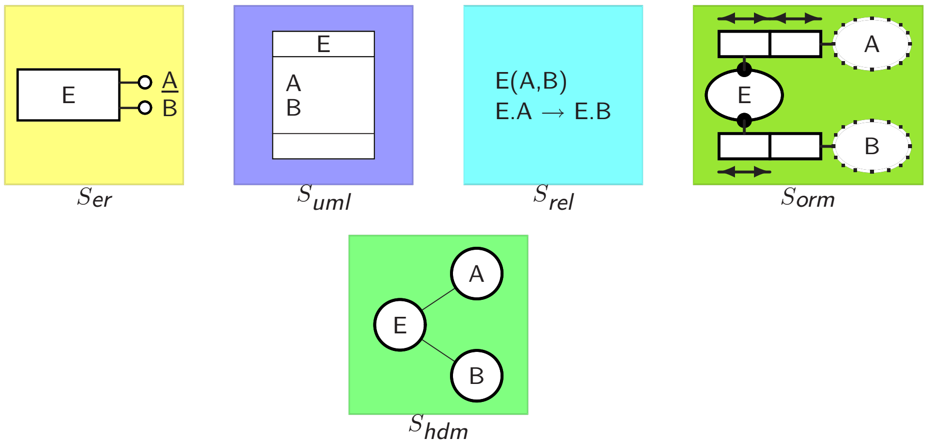


Figure 1: Conceptual modelling languages represented in the HDM

In [15] a general approach was proposed showing how the data aspects of the higher level modelling languages were modelled as nodes and edges in the HDM, with the constraints of the higher level modelling language being represented by writing constraint formulae over the HDM. This approach was implemented in AutoMed [2]. This paper extends that approach to represent the high level modelling language constraints using a proposed set of primitive constraint operators on the HDM. This paper also shows how we may relate the ER, relational, UML and ORM higher level modelling languages – perform **intermodel transformations** – by the application of five types of equivalence rules on the HDM and its primitive constraint operators. This work is a considerable enhancement of our earlier work presented in [3].

To date, work on intermodel transformations has normally defined the conversion between specific pairs of modelling languages. For example, there have been proposals for relational to ER conversion [1, 19], ORM to UML and relational conversion [11], and relational to generic object oriented conversion [6]. Our approach differs from this previous work in that it uses the semantic definition of each modelling language in the HDM as a basis for equivalence rules on the HDM to perform the conversion. Thus we provide a platform for the conversion between any data modelling language, provided that we can represent that modelling language in the HDM with the proposed primitive constraints defined in this paper. This paper demonstrates the approach being applied by converting between the major constructs of ER, relational, UML class and ORM modelling languages.

In addition to providing a mechanism for comparing the expressiveness of modelling languages, the proposed primitive constraints and set of equivalence rules also forms the basis for a method of automating the translation between modelling languages, based on descriptions of their constructs. This would involve further development of an algorithm that would determine which equivalence rules need to be applied to the HDM graph of one higher level model to form a valid HDM graph of another higher level model.

The paper is structured as follows. Section 2 reviews how to describe higher level data modelling languages by relating them to the graph structure. The graph language is extended with a set of constraint operators that form a language used to model the constraints in higher level data modelling languages. Section 3 details how we approach the transformation between modelling languages by applying equivalence rules to the graph, thereby relating basic constructs of the higher level modelling languages with each other. Section 4 considers how some extended operators of these languages are related.

2 Describing a Data Modelling Language

When modelling a data model in HDM one is often faced with choices when deciding how to represent the data model with a set of constructs. Ideally the chosen constructs will map directly to the data model's constructs and constrain the construction process such that only valid schemas of the data model can be constructed.

In [15] a general technique was proposed for the modelling of any structured data modelling language in the HDM. The premise of this approach is that in any data modelling language, the various constructs of the language can be viewed as being a combination of sets of values, and constraints between those sets, in a graph based model. This concept has been used in modelling relational models [24], and for OO and ER models [23], and we argue is an intuitive assumption to make. It also reduces all models to an irreducible form [10], and in the context of relational databases has recently been identified as a sixth normal form [8, 7].

A HDM model M consists of a tuple $\langle Nodes, Edges, Cons\rangle$, 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*. A **hyperedge** is an edge that connects more than two nodes in a graph, and a **nested edge** is one which connects to another edge rather than just nodes. When used to describe a data source, each node has an **extent** that represents the set of values from the data source that are associated with the node, and each edge also has an extent, where the values the edge extent contains must also appear in the extent of the nodes and edges that the edge connects. Details of the HDM are found in [21]. The HDM is a simplification of the hypergraph model in [20], which allowed nodes to contain complete graphs.

We now introduce to the HDM a set of six primitive constraints that may be used to model the constraints of the higher level modelling language in an analogous manner to how the nodes and edges of the HDM model other features of the higher level modelling language. In the following descriptions, N denotes any node, E any edge, NE any node or edge, and $\vec{NE}$ any set of nodes or edges.

- **inclusion** $\vec{NE}_1 \subseteq \vec{NE}_2$: The extent of tuples in $\vec{NE}_1$ is a subset of the extent of tuples $\vec{NE}_2$. If $\vec{NE}$ is more that one construct, then the various members of the set must be related via some edge.
- **exclusion** $\not\bigcap(NE_1 \ldots NE_n)$: For every x, y for which $1 \leq x < y \leq n$, the extent of NE_x does not intersect with the extent of NE_y.
- **union** $NE = \bigcup \vec{NE}_s$: The extent of NE is the union of the extents of $\vec{NE}_s$.
- **mandatory** $\vec{NE} \overset{n}{\rhd} E$: every node or edge in $\vec{NE}$ is connected by edge E, and every combination of instances in the extents of $\vec{NE}$ must appear at least n times in the extent of E.
- **unique** $\vec{NE} \overset{n}{\lhd} E$: every node or edge of $\vec{NE}$ is connected by edge E, every combination of instances in the extents of $\vec{NE}$ must appear no more than n times in the extent of E.
- **reflexive** $\vec{NE} \xrightarrow{\text{id}} E$: If an instance of $\vec{NE}$ appears in E, then one of the instances of E must be an identity tuple: *i.e.* if $\langle a_1, \ldots, a_m\rangle$ of NE appears in a tuple of E, then one of those tuples in E is $\langle\langle a_1, \ldots, a_m\rangle, a_1, \ldots, a_m\rangle$.

We will abbreviate $\overset{1}{\rhd}$ as $\rhd$ and $\overset{1}{\lhd}$ as $\lhd$. Combinations of these constraints may be used to represent constraints in the higher level modelling language. For example, **cardinality constraints** may be represented by a combination of mandatory and unique as follows:

None	$\rightarrow$	NE has 0:N occurrences in E
$NE \rhd E$	$\rightarrow$	NE has 1:N occurrences in E
$NE \lhd E$	$\rightarrow$	NE has 0:1 occurrences in E
$NE \rhd E \wedge NE \lhd E$	$\rightarrow$	NE has 1:1 occurrences in E

Most of these constraint operators have been used before in the context of modelling single modelling languages. In particular, mandatory and unique constraints have been used in a hypergraph model for relational schemas in [24], and inclusion constraints appear in [22]. The appearance of the reflexive constraint might be surprising. However, reflexive in combination with mandatory and unique, allows for the description of natural keys to be used in a data modelling language. For example, the relational model in Figure 3(a) uses $\rhd$, $\lhd$ and $\xrightarrow{\text{id}}$ between node ⟨⟨student⟩⟩ (representing the table student) and edge ⟨⟨_,student,student:name⟩⟩ to state that ⟨⟨student:name⟩⟩ (representing the column name) is the key for ⟨⟨student⟩⟩. This is because the combination of $\rhd$ and $\lhd$ mean there is only one instance in the edge's extent for each instance of ⟨⟨student⟩⟩, and the $\xrightarrow{\text{id}}$ forces the value of ⟨⟨student:name⟩⟩ to be the same since the edge is reflexive.

It should be also noted that [23] argues for simplicity in modelling languages, and so using the simple HDM as the common data modelling language is an approach that has been argued for before. However, it should be noted that what we are arguing for in this paper is to use the HDM as a method for comparing and transforming between various other data modelling languages, and *not* for using HDM as a modelling language for new applications.

Figures 2(a), 3(a), 4(a) and 5(a) show four data models, in ER, relational, UML and ORM data modelling languages. These are designed to cover the same **universe of discourse** (**UoD**), and as will be shown later, three of them have the same information capacity [17]. The schemas represent a record of students, the courses that they sit, and the grades they obtain for those courses. Some students are undergraduates, and each ug has an associated personal programming tutor ppt that other students do not have. The use of underlining in the relational and ER models indicates what are key attributes, and a question mark follows a nullable attribute in those models. In the relational model, foreign keys are shown by using an implication between the foreign key columns and the referenced table columns. In the ER model this foreign key may either be represented by a relationship (for example the foreign keys result.name → student.name and result.code → course.code are represented in the ER result relationship) or by a subset (for example the foreign key ug.name → student.name is represented by a subset between the student and ug entities).

2.1 Describing an ER Model Language in the HDM

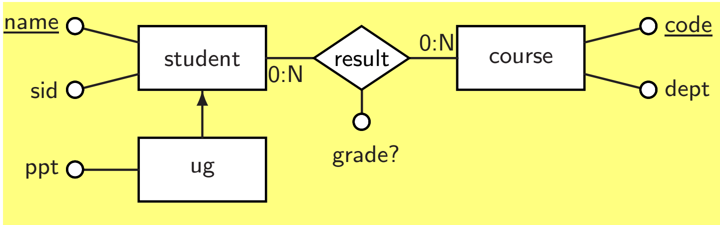
When the HDM is used to model a higher level modelling language, each construct in that language must be classified as being one of four types, each of which imply a different representation in the HDM. We explain how this methodology (first presented in [15]) is applied to an ER modelling language (which we describe here, see [18] for a survey of variations of ER modelling languages), and illustrate our discussions by showing how the methodology may take the ER model of Figure 2(a) and produce the HDM model of Figure 2(b). Note that in the HDM diagrams, HDM nodes are represented by white circles with thick outlines, and HDM edges are represented by thick black lines. The HDM constraint language is represented by grey dashed boxes connected by grey lines to the nodes and edges to which the constraint applies.

2.1.1 Nodal

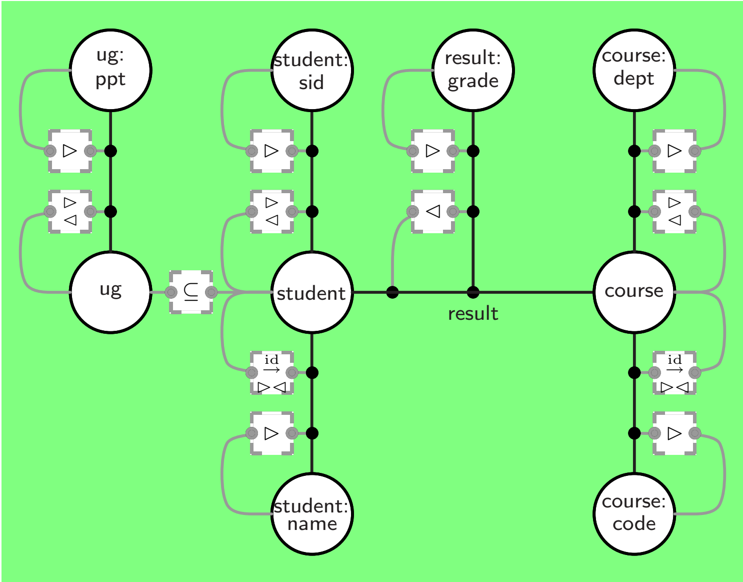
A **nodal** construct is one that may appear in isolation in a model, such as an ER model **entity**. Using the AutoMed data integration system [2], such constructs are defined by giving a prototype **scheme** that must contain the name of a HDM node used to represent that construct. Hence we represent the ER entity student by the schema ⟨⟨student⟩⟩. In this paper we introduce a method to translate the higher level constructs in the HDM by use of simple production rules. The rules give a pattern of a higher level modelling language scheme, which when matched, will produce the HDM nodes and edges listed after the $\rightsquigarrow$ symbol. There may follow a number of guarded auxiliary rules.

The production rule for an ER entity $\langle\langle E\rangle\rangle$ is very simple, since it states that each entity with scheme $\langle\langle E\rangle\rangle$ maps to a single HDM node $\langle\langle E\rangle\rangle$, and has no constraints:

entity $\langle\langle E\rangle\rangle \rightsquigarrow \langle\langle E\rangle\rangle$



(a) An ER model of the student-course database



(b) HDM representation of the ER model

Figure 2: An ER model and its equivalent HDM model

2.1.2 Link

A **link** construct is one that associates other constructs with each other, and which has an extent which is drawn from those constructs, such as an ER **relationship** construct. In AutoMed, we represent ER relationships by the scheme made up of the of the name of the HDM edge used to represent the construct, together with pairs of the entity names and cardinality constraints. For example, we represent the ER relationship result in Figure 2(a) by the scheme $\langle\langle \mathsf{result}, \mathsf{student}, \mathsf{0{:}N}, \mathsf{course}, \mathsf{0{:}N}\rangle\rangle$. The production rule uses auxiliary rules to generate the constraints in the HDM necessary to represent the cardinality constraints in the ER model.

$\mathsf{relationship}\ \langle\langle R, E_1, L_1{:}U_1, \ldots, E_n, L_n{:}U_n\rangle\rangle \leadsto \langle\langle R, E_1, \ldots, E_n\rangle\rangle$

$\quad \mathsf{true} \quad \rightarrow \mathsf{generate\_card}(E_1, \langle\langle R, E_1, \ldots, E_n\rangle\rangle, L_1, U_1)$

$\quad\quad\quad \vdots$

$\quad \mathsf{true} \quad \rightarrow \mathsf{generate\_card}(E_n, \langle\langle R, E_1, \ldots, E_n\rangle\rangle, L_n, U_n)$

The cardinality constraints are produced by a function which shall be used for all the modelling languages we consider, and translates their cardinality constraints into applications of $\triangleright$ and $\triangleleft$ in the HDM.

$\mathsf{generate\_card}\ (NE, E, L{:}U) \rightsquigarrow \bot$

$L > \mathsf{0} \quad \rightarrow NE \overset{L}{\rhd} E$

$U < \mathsf{*} \quad \rightarrow NE \overset{U}{\lhd} E$

Thus the production rule when applied to ⟨⟨result, student, 0:N, course, 0:N⟩⟩ produces an edge ⟨⟨_, result, student⟩⟩ to represent its extent. The auxiliary constraint rules will produce no constraints, since neither of the guards within the definition of generate_id will match $L = \mathsf{0}$ or $U = \mathsf{*}$.

2.1.3 Link-Nodal

A **link-nodal** construct is one that has associated values, 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 name attribute of the entity student is represented in AutoMed by the scheme ⟨⟨student, name, notnull⟩⟩. The production rule for ER attributes creates a node and edge, the last construct listed after the $\rightsquigarrow$ being the extent of the higher level rule.

$\mathsf{attribute}\ \langle\langle E, A, N\rangle\rangle \rightsquigarrow \langle\langle E{:}A\rangle\rangle, \langle\langle \_, E, E{:}A\rangle\rangle$

$\mathsf{true} \quad \rightarrow \mathsf{generate\_card}(\langle\langle E{:}A\rangle\rangle, \langle\langle \_, E, E{:}A\rangle\rangle, \mathsf{1}, \mathsf{*})$

$N = \mathsf{notnull} \quad \rightarrow \mathsf{generate\_card}(\langle\langle E\rangle\rangle, \langle\langle \_, E, E{:}A\rangle\rangle, \mathsf{1}, \mathsf{1})$

$N = \mathsf{null} \quad \rightarrow \mathsf{generate\_card}(\langle\langle E\rangle\rangle, \langle\langle \_, E, E{:}A\rangle\rangle, \mathsf{0}, \mathsf{1})$

Thus the production rule when applied to the ER attribute ⟨⟨course,dept,notnull⟩⟩ produces the node ⟨⟨course:dept⟩⟩ and the edge ⟨⟨_,course,course:dept⟩⟩ to represent the extent of the attribute. The first auxiliary constraint rule produces ⟨⟨course:dept⟩⟩ $\rhd$ ⟨⟨_,course,course:dept⟩⟩, the second produces ⟨⟨course⟩⟩ $\rhd$ ⟨⟨_,course,course:dept⟩⟩ and ⟨⟨course⟩⟩ $\lhd$ ⟨⟨_,course,course:dept⟩⟩ (since both guards in the generate_card are met), and then the last rule fails to match in the guard.

Note that since the grade attribute is optional, we obtain just two constraints when the production rule is used on ⟨⟨result,grade,null⟩⟩:

⟨⟨result:grade⟩⟩ $\rhd$ ⟨⟨_,⟨⟨result,student,course⟩⟩,result:grade⟩⟩

⟨⟨result,student,course⟩⟩ $\lhd$ ⟨⟨_,⟨⟨result,student,course⟩⟩,result:grade⟩⟩

Note that in our modelling of the ER model (and relational and UML languages), the fact that attribute names are prefixed by the associated entity name reflects a deliberate choice made when defining the construct. One could alternatively say that attribute names are globally unique, which would change the HDM graph to have just one node ⟨⟨name⟩⟩ to represent both the ⟨⟨student, name, notnull⟩⟩ and ⟨⟨ug, name, notnull⟩⟩ relational columns, but this would not give the correct semantics for a normal ER model. The alternative global naming choice will be used in modelling the value types of ORM models.

In Figure 2(b) it should be noted that the syntax is not ambiguous, but needs careful reading. Each $\rhd$ or $\lhd$ always has a node or edge on its left hand side that appears in the edge on the right hand side. We use this fact to ignore which 'side' we connect $\rhd$ and $\lhd$ constraints to in the diagram. This makes the diagrams more tidy in appearance. (Note that this is different from the approach we followed in our earlier work [3]). Therefore the ⟨⟨course:dept⟩⟩ to ⟨⟨_,course,course:dept⟩⟩ mandatory constraint is drawn using $\rhd$ in the constraint box.

2.1.4 Constraint

A **constraint** construct is one that has no associated extent, but instead limits the extent of the constructs it connects to. An example of a constraint construct is the ER model **subset** relationship. For example, the subset between ug and student is represented in AutoMed by the scheme ⟨⟨student, ug⟩⟩.

$\mathsf{subset}\ \langle\langle E, E_s\rangle\rangle \rightsquigarrow \bot$

$\mathsf{true} \quad \rightarrow \langle\langle E_s\rangle\rangle \subseteq \langle\langle E\rangle\rangle$

ER **generalisations** are another example of constraint constructs. For example, a generalisation that specified the children entities $\langle\langle E_1\rangle\rangle, \ldots, \langle\langle E_n\rangle\rangle$ are disjoint subsets of some parent entity $\langle\langle E\rangle\rangle$ could be defined by the following rule:

generalisation $\langle\langle E, E_1, \ldots, E_n \rangle\rangle \rightsquigarrow \bot$
  true $\rightarrow \langle\langle E_1 \rangle\rangle \subseteq \langle\langle E \rangle\rangle, \ldots, \langle\langle E_n \rangle\rangle \subseteq \langle\langle E \rangle\rangle$
  true $\rightarrow \not\bowtie(\langle\langle E_1 \rangle\rangle, \ldots, \langle\langle E_n \rangle\rangle)$

Our example ER model in Figure 2(a) contains no generalisations, but we will discuss and compare advanced modelling constructs of various data modelling languages in Section 4.

The final constraint in our ER model is the definition of the **key** of an entity, which servers to denote the set of its attributes that may be used to identify instances of the attribute.

key $\langle\langle E, A_1 \rangle\rangle, \ldots, \langle\langle E, A_n \rangle\rangle \rightsquigarrow \bot$
  true $\rightarrow \langle\langle E \rangle\rangle \xrightarrow{\text{id}} \bowtie (\langle\langle \_, E, E{:}A_1 \rangle\rangle, \ldots, \langle\langle \_, E, E{:}A_n \rangle\rangle)$

The constraint limits the instances of the entity to be an identity with the join of its key attributes (we will give an example of how this type of constraint works when looking at the relational result table in the next section).

2.2 Describing the relational model in the HDM

ug	
name	ppt
Mary	NR
Jane	SK

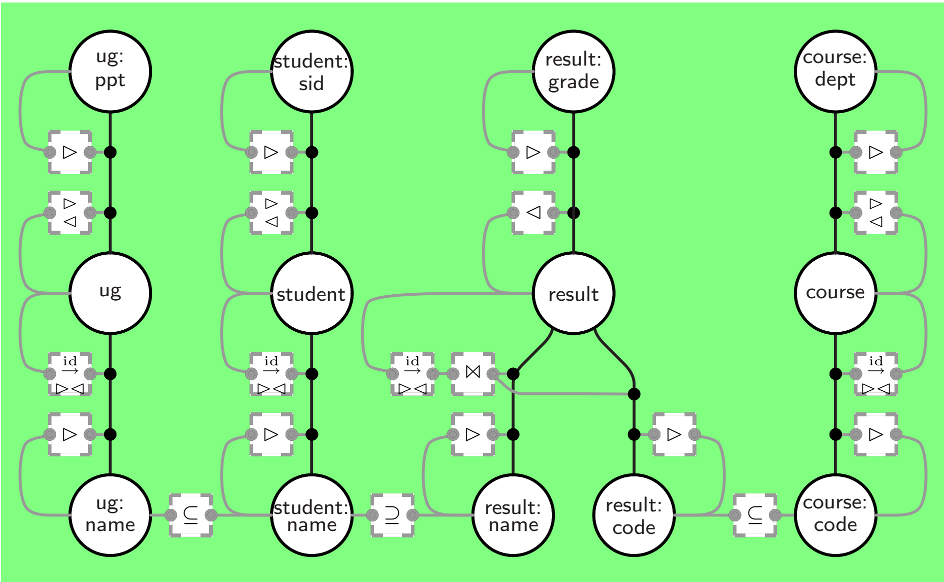
student	
name	sid
Mary	1
John	2
Jane	3
Fred	4

course	
code	dept
DB	CS
Fin	CS
Geo	Maths

result		
code	name	grade?
DB	Mary	A
Fin	Jane	C
Fin	Fred	null
Geo	Fred	A
Geo	John	B

ug.name $\rightarrow$ student.name
result.name $\rightarrow$ student.name
result.code $\rightarrow$ course.code

(a) Relational database schema and data



(b) HDM representation of relational database schema

Figure 3: A relational model for the student-course database

Having reviewed the general methodology for representing higher level modelling languages in the HDM in the previous subsection, we will now apply the methodology to the relational model. Relational model **table**s are nodal constructs, and hence we represent the table student by the scheme $\langle\langle$student$\rangle\rangle$, and the table result by $\langle\langle$result$\rangle\rangle$. The production rule for translating such schemes into the HDM is as follows.

table $\langle\langle T\rangle\rangle \rightsquigarrow \langle\langle T\rangle\rangle$

Relational model **column**s are link-nodal constructs, and hence are modelled by a scheme containing a HDM node that represents the construct it depends on, followed by the name of the HDM node that represents the column, followed by the constraint on whether the attribute may be null. For example, the name column of table student is represented by the scheme $\langle\langle$student, name, notnull$\rangle\rangle$. In the HDM, this becomes a node $\langle\langle$student:name$\rangle\rangle$ to represent values of the column/attribute, and the nameless edge $\langle\langle$_,student,student:name$\rangle\rangle$ to represent the association of these values to table/entity $\langle\langle$student$\rangle\rangle$.

column $\langle\langle T, C, N\rangle\rangle \rightsquigarrow \langle\langle T{:}C\rangle\rangle, \langle\langle \_, T, T{:}C\rangle\rangle$

true $\rightarrow \mathsf{generate\_card}(\langle\langle T{:}C\rangle\rangle, \langle\langle \_, T, T{:}C\rangle\rangle, 1, *)$

$N = \mathsf{notnull} \rightarrow \mathsf{generate\_card}(\langle\langle T\rangle\rangle, \langle\langle \_, T, T{:}C\rangle\rangle, 1, 1)$

$N = \mathsf{null} \rightarrow \mathsf{generate\_card}(\langle\langle T\rangle\rangle, \langle\langle \_, T, T{:}C\rangle\rangle, 0, 1)$

The definition of relational columns and ER attributes are very similar, and as can be seen by comparing Figures 3 and 2, produce similar results in the HDM.

The **primary key** construct of the relational modal is a constraint construct. The constraint specifies that the natural join between its key columns gives the extent of the table. The schema of a constraint simply needs to list the table and the columns. For example, the primary key of $\langle\langle$result$\rangle\rangle$ would be represented in the HDM by $\langle\langle$result$\rangle\rangle \xrightarrow{\text{id}} (\langle\langle$_,result,result:code$\rangle\rangle \bowtie \langle\langle$_,result,result:name$\rangle\rangle)$. The production rule to produce the HDM constraints is as follows:

primary_key $\langle\langle T, C_1, \ldots, C_n\rangle\rangle \rightsquigarrow \perp$

true $\rightarrow \langle\langle T\rangle\rangle \xrightarrow{\text{id}} \bowtie (\langle\langle \_, T, T{:}C_1\rangle\rangle, \ldots, \langle\langle \_, T, T{:}C_n\rangle\rangle)$

Since any key column must also be a notnull column in a valid relational model, this rule need only add the fact that the join of the key columns is reflexive. Since each column is individually mandatory and unique w.r.t the table, it follows that the join is also mandatory and unique, as shown for the result node in Figure 3(b). For the primary key scheme $\langle\langle$result,name,code$\rangle\rangle$ for the result table, the production rule generates the reflexive constraint $\langle\langle$result$\rangle\rangle \xrightarrow{\text{id}} \bowtie (\langle\langle$_,result,result:code$\rangle\rangle$, $\langle\langle$_,result,result:name$\rangle\rangle)$. For example with the relation data shown in Figure 3(a), this constraint along with the already stated mandatory and unique constraints enforce the following type of instantiation of the $\langle\langle$result$\rangle\rangle$ node and key edges:

$\langle\langle$result$\rangle\rangle$ = [{DB, Mary}, {Fin, Jane}, {Fin, Fred}, ...]

$\langle\langle$_,result,result:name$\rangle\rangle$ =

[{{DB, Mary}, Mary}, {{Fin, Jane}, Jane}, {{Fin, Fred}, Fred}, ...]

$\langle\langle$_,result,result:code$\rangle\rangle$ =

[{{DB, Mary}, DB}, {{Fin, Jane}, Fin}, {{Fin, Fred}, Fin}, ...]

We represent the **foreign key** constraint by the scheme made up of a name for the constraint, the table and column(s) that are the foreign key, and the table and column(s) of the referenced table.

foreign_key $\langle\langle FK, T, C_1, \ldots, C_n, T_f, C_{f_1}, \ldots, C_{f_n}\rangle\rangle \rightsquigarrow \perp$

true $\rightarrow \pi_{\langle\langle T:C_1\rangle\rangle,\ldots,\langle\langle T:C_n\rangle\rangle} \bowtie (\langle\langle \_, T, T{:}C_1\rangle\rangle, \ldots, \langle\langle \_, T, T{:}C_n\rangle\rangle) \subseteq$

$\pi_{\langle\langle T_f:C_{f_1}\rangle\rangle,\ldots,\langle\langle T_f:C_{f_n}\rangle\rangle} \bowtie (\langle\langle \_, T_f, T_f{:}C_{f_1}\rangle\rangle, \ldots, \langle\langle \_, T_f, T_f{:}C_{f_1}\rangle\rangle)$

The somewhat complex constraint simply states that the join of the columns listed in T is a subset of the join of the columns in T_f. For the common case where foreign keys are not compound keys (*i.e.* $n = 1$), the constraint would simplify to $\langle\langle T{:}C_1\rangle\rangle \subseteq \langle\langle T_f{:}C_{f_1}\rangle\rangle$ For example, the foreign key between ug and student is represented by the scheme $\langle\langle$ug_fk, ug, name, student, name$\rangle\rangle$. Using the production rule, this scheme becomes $\langle\langle$ug:name$\rangle\rangle \subseteq \langle\langle$student:name$\rangle\rangle$ in the HDM.

Finally, the relational **candidate key** takes a similar definition to primary key, except that all that is established is a mandatory and a unique association between the table and a join of the candidate key columns.

$\mathsf{candidate\_key}\ \langle\!\langle T, C_1, \ldots, C_n \rangle\!\rangle \leadsto \bot$

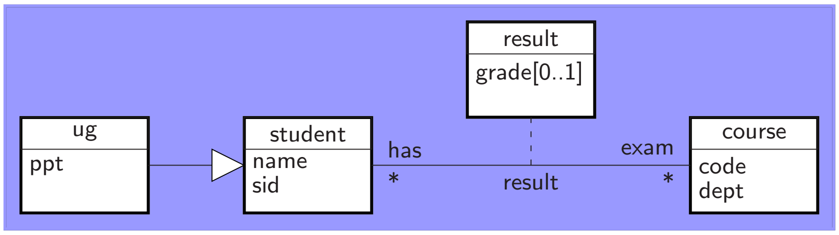
$\mathsf{true} \quad \rightarrow \langle\!\langle T \rangle\!\rangle \lhd \bowtie (\langle\!\langle \_, T, T{:}C_1 \rangle\!\rangle, \ldots, \langle\!\langle \_, T, T{:}C_n \rangle\!\rangle)$

$\mathsf{true} \quad \rightarrow \langle\!\langle T \rangle\!\rangle \rhd \bowtie (\langle\!\langle \_, T, T{:}C_1 \rangle\!\rangle, \ldots, \langle\!\langle \_, T, T{:}C_n \rangle\!\rangle)$

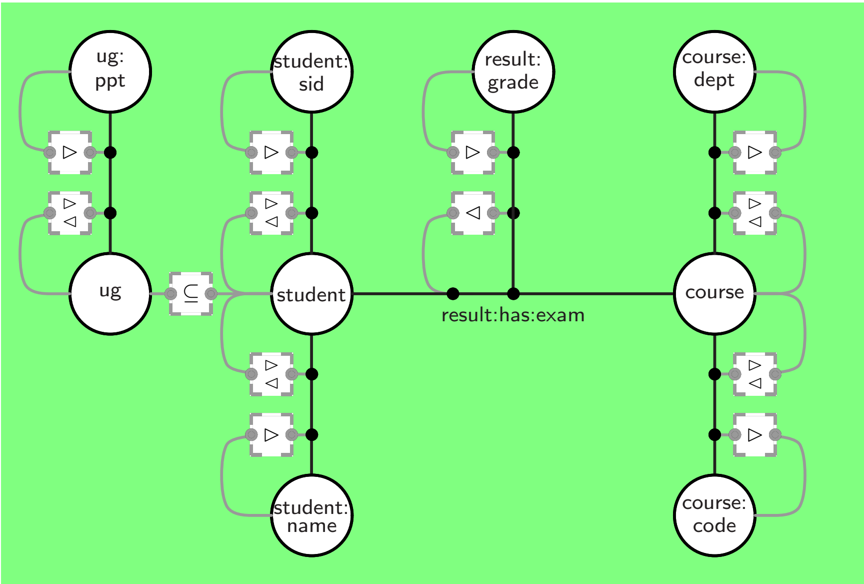
$\mathsf{true} \quad \rightarrow (\pi_{\langle\!\langle T:C_1 \rangle\!\rangle, \ldots, \langle\!\langle T:C_n \rangle\!\rangle} \bowtie (\langle\!\langle \_, T, T{:}C_1 \rangle\!\rangle, \ldots, \langle\!\langle \_, T, T{:}C_n \rangle\!\rangle)) \lhd \bowtie (\langle\!\langle \_, T, T{:}C_1 \rangle\!\rangle, \ldots, \langle\!\langle \_, T, T{:}C_n \rangle\!\rangle)$

The last line ensures that the combination of columns in the candidate key appears just once in the edge formed by the join of the candidate key column edges. In the common case where the candidate key is not compound (*i.e.* $n = 1$), the last constraint simplifies to $\langle\!\langle T{:}C_1 \rangle\!\rangle \lhd \langle\!\langle \_, T, T{:}C_1 \rangle\!\rangle$. Thus if we added the new candidate key $\langle\!\langle \mathsf{student,sid} \rangle\!\rangle$ to the example relational model, then we would add to our existing relation HDM a $\langle\!\langle \mathsf{student{:}sid} \rangle\!\rangle \lhd \langle\!\langle \mathsf{\_,student,student{:}sid} \rangle\!\rangle$ constraint.

2.3 Describing UML in the HDM



(a) A UML class model of the student-course database



(b) HDM representation of the UML Model

Figure 4: A UML model and its equivalent HDM model

UML **classes** are nodal constructs, and hence each UML class scheme $\langle\!\langle C \rangle\!\rangle$ maps to a single node $\langle\!\langle C \rangle\!\rangle$. The extent of $\langle\!\langle C \rangle\!\rangle$ is the set of unique **object identifiers** (**OID**) of the class.

$\mathsf{class}\ \langle\!\langle C \rangle\!\rangle \leadsto \langle\!\langle C \rangle\!\rangle$

The definition of n-ary **associations** in UML states that the multiplicity, $L..U$, of a role, R, defines the number of instances of the class C that are associated with a particular set of values of the other classes in the association A. Hence the generation of constraints for UML associations takes all the classes except the role class when calling $\mathsf{generate\_card}$ on a role. We also make the assumption that $*$ is simply a shorthand for $\mathsf{0..*}$, and any single number n is a shorthand for $\mathsf{n..n}$.

$$\begin{array}{ll} \multicolumn{2}{l}{\mathsf{association}\ \langle\!\langle A, R_1, C_1, L_1..U_1, \ldots, R_n, C_n, L_n..U_n \rangle\!\rangle \leadsto \langle\!\langle A, C_1, \ldots, C_n \rangle\!\rangle} \\ \quad \mathsf{true} & \rightarrow \mathsf{generate\_card}(\{C_2, \ldots, C_n\}, \langle\!\langle A{:}R_1{:}\ldots{:}R_n, C_1, \ldots, C_n \rangle\!\rangle, L_1, U_1) \\ & \vdots \\ \quad \mathsf{true} & \rightarrow \mathsf{generate\_card}(\{C_1, \ldots, C_{n-1}\}, \langle\!\langle A{:}R_1{:}\ldots{:}R_n, C_1, \ldots, C_n \rangle\!\rangle, L_n, U_n) \end{array}$$

For example, the scheme $\langle\!\langle$result,has,student,0..*,exam,course,0..*$\rangle\!\rangle$ models the UML association between student and course, and the production rule maps this scheme to the HDM edge $\langle\!\langle$result:has:exam,student,course$\rangle\!\rangle$, with no constraints. Note that the label of the HDM edge is $A{:}R_1{:}\ldots{:}R_n$, which encodes the various labels HDM gives the association in a single HDM identifier. Thus the result association with role names has and exam gets the HDM edge name result:has:exam.

UML **attributes** are link-nodal constructs attached to the UML class, and hence the production rule takes a similar form to that for ER attributes or relational columns. We make the same assumptions about shorthands for the attribute multiplicity as we did for association multiplicity, as well as noting that the absence of explicit multiplicity means that 1..1 is assumed. Thus the sid attribute of student has the scheme $\langle\!\langle$student,sid,1..1$\rangle\!\rangle$.

$$\begin{array}{ll} \multicolumn{2}{l}{\mathsf{attribute}\ \langle\!\langle C, A, L..U \rangle\!\rangle \leadsto \langle\!\langle C{:}A \rangle\!\rangle, \langle\!\langle \_, C, C{:}A \rangle\!\rangle} \\ \quad \mathsf{true} & \rightarrow \mathsf{generate\_card}(\langle\!\langle C{:}A \rangle\!\rangle, \langle\!\langle \_, C, C{:}A \rangle\!\rangle, 1, *) \\ \quad L..U & \rightarrow \mathsf{generate\_card}(\langle\!\langle C \rangle\!\rangle, \langle\!\langle \_, C, C{:}A \rangle\!\rangle, L, U) \end{array}$$

UML **generalisations** have a sophisticated constraint system that specifies that the various classes or associations that are children of a parent class or association are overlapping, disjoint, complete or incomplete. The first and last of these keywords are 'noise' in the sense that they add nothing in addition to an unlabelled generalisation. The other two add an exclusion constraint and a union constraint.

$$\begin{array}{ll} \multicolumn{2}{l}{\mathsf{generalisation}\ \langle\!\langle C, C_1, \ldots, C_n, D \rangle\!\rangle \leadsto \bot} \\ \quad \mathsf{true} & \rightarrow \langle\!\langle C_1 \rangle\!\rangle \subseteq \langle\!\langle C \rangle\!\rangle, \ldots, \langle\!\langle C_n \rangle\!\rangle \subseteq \langle\!\langle C \rangle\!\rangle \\ \quad \mathsf{disjoint} \in D & \rightarrow \not\cap(\langle\!\langle C_1 \rangle\!\rangle, \ldots, \langle\!\langle C_n \rangle\!\rangle) \\ \quad \mathsf{complete} \in D & \rightarrow \langle\!\langle C \rangle\!\rangle = \bigcup(\langle\!\langle C_1 \rangle\!\rangle \ldots \langle\!\langle C_n \rangle\!\rangle) \end{array}$$

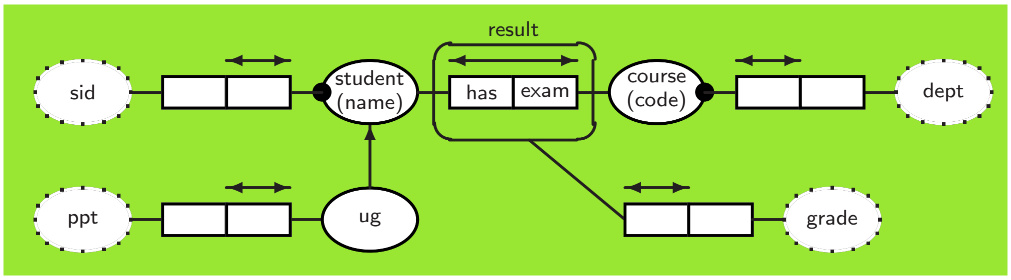
2.4 Describing the ORM in the HDM

From our analysis of the ER, relational and UML modelling languages, it may seem 'obvious' that ORM entity types should be modelled as nodal constructs while value types should be modelled as link-nodal constructs. However, due to ORM's rich semantics, the similarity of value type and entity type roles in fact types, and a value-type's ability to play multiple roles in fact types, it is correct to model both value types and entity types using a single HDM nodal construct type.

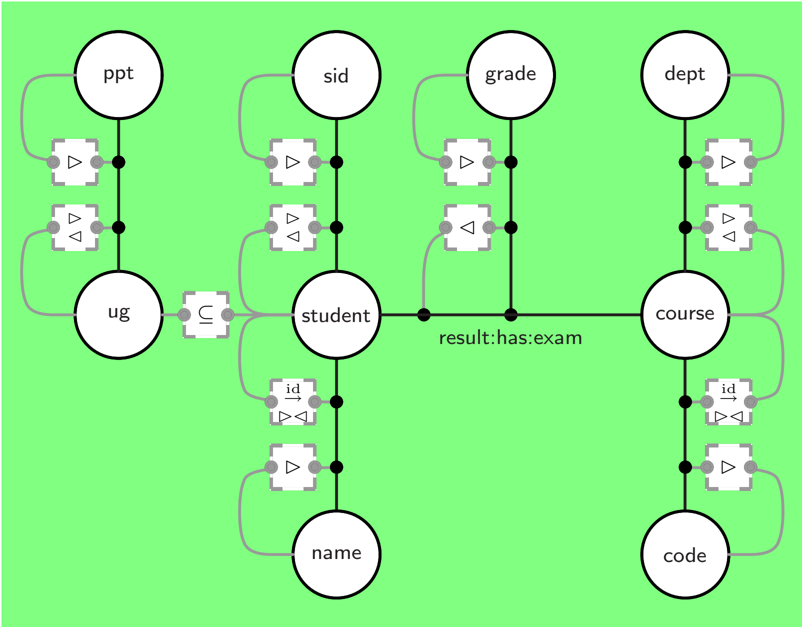
Each **entity type** $\langle\!\langle E \rangle\!\rangle$ maps to a single node $\langle\!\langle E \rangle\!\rangle$. The extent of entity type $\langle\!\langle E \rangle\!\rangle$ is the extent of its primary reference mode while the extent of a **value type** $\langle\!\langle V \rangle\!\rangle$ is just the ORM value type's set of values. Hence we have the simple definitions:

$$\begin{array}{l} \mathsf{entity\_type}\ \langle\!\langle E \rangle\!\rangle \leadsto \langle\!\langle E \rangle\!\rangle \\ \mathsf{value\_type}\ \langle\!\langle V \rangle\!\rangle \leadsto \langle\!\langle V \rangle\!\rangle \end{array}$$

An ORM n-ary **fact type** is an association between n objects where each object is an entity type, value type, or objectified fact type. A fact type's extent is drawn from the objects it associates and is hence modelled in the HDM as a link construct. The scheme for the ORM fact type should describe the name of the fact type (if any) along with the role name (if any), role object, and the mandatory nature of the object type in the role. Hence the fact type between $\langle\!\langle$student$\rangle\!\rangle$ and $\langle\!\langle$course$\rangle\!\rangle$ has the scheme $\langle\!\langle$result,has,student,_,exam,course,_$\rangle\!\rangle$ and the fact type between $\langle\!\langle$student$\rangle\!\rangle$ and $\langle\!\langle$sid$\rangle\!\rangle$ has scheme $\langle\!\langle$_,student,_,●,sid,_,_$\rangle\!\rangle$. These then map into the HDM using the following production rule:



(a) An ORM model of the student-course database



(b) HDM representation of the ORM model

Figure 5: An ORM model of the student-course database

$\mathsf{fact\_type}\ \langle\!\langle FT, N_1, R_1, M_1, \ldots, N_n, R_n, M_n \rangle\!\rangle \leadsto \langle\!\langle \langle\!\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n \rangle\!\rangle \rangle\!\rangle$

$M_1 = \bullet \quad \rightarrow \mathsf{generate\_card}(R_1, \langle\!\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n \rangle\!\rangle, 1, *)$

$\vdots$

$M_n = \bullet \quad \rightarrow \mathsf{generate\_card}(R_n, \langle\!\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n \rangle\!\rangle, 1, *)$

The names of fact types and roles are encoded into a single HDM edge label in a similar manner to that used to encode UML association and role names into a single HDM edge label. Note that all fact types have an implied uniqueness constraint across all n roles, and HDM has a similar edge constraint because the extent of an edge is a set of tuples. In ORM one can specify uniqueness constraints across $n-1$ roles of an n role fact type. Hence we define the scheme of **uniqueness** to take both a fact type and the role that is uniquely identified by the other roles:

$\mathsf{uniqueness}\ \langle\!\langle \langle\!\langle FT, N_1, R_1, M_1, \ldots, N_n, R_n, M_n \rangle\!\rangle, R_x \rangle\!\rangle \leadsto \bot$

$\mathsf{true} \quad \rightarrow \mathsf{generate\_card}(\{R_1, \ldots, R_{x-1}, R_{x+1}, \ldots, R_n\},$
$\qquad \langle\!\langle \langle\!\langle FT{:}N_1{:}\ldots{:}N_n, R_1, \ldots, R_n \rangle\!\rangle \rangle\!\rangle, 0, 1\ )$

We note in passing that ORM also has a general **frequency constraint** type across any number of roles. We have not needed this in our examples, but could have modelled the mandatory and unique constraints using the more general frequency constraint; but as there are implicit unique and mandatory constraints in an ORM schema and these constraints are used heavily in the rules regarding a schema's well formedness, it is useful to model mandatory and unique as we have here.

ORM can express **subtype** relationships between fact roles as well as entity types. We only use subtyping between entity types in our examples, hence we shall restrict ourselves to just defining that below, together with the notion of disjointness and totality of such subtypes which ORM also supports:

subset $\langle\langle EV, EV_s\rangle\rangle \rightsquigarrow \bot$
true $\rightarrow \langle\langle EV_s\rangle\rangle \subseteq \langle\langle EV\rangle\rangle$
disjoint $\langle\langle EV_1\rangle\rangle, \langle\langle EV_2\rangle\rangle \rightsquigarrow \bot$
true $\rightarrow \langle\langle EV_1\rangle\rangle \not\bowtie \langle\langle EV_2\rangle\rangle$
total $\langle\langle EV, EV_1\rangle\rangle, \langle\langle EV, EV_2\rangle\rangle \rightsquigarrow \bot$
true $\rightarrow \langle\langle EV\rangle\rangle \cup \{\langle\langle EV_1\rangle\rangle, \langle\langle EV_2\rangle\rangle\}$

Applying the above definitions to the ORM diagram in Figure 5(a) produces the HDM in Figure 5(b), almost same HDM we arrived at using the ER model bar some trivial renaming of nodes and edges. Note that we have assumed that the value classes implied by the primary reference modes for each entity class have been made explicit before applying the definitions.

3 Inter Model Transformations

We now introduce five general purpose equivalence mappings that may be used on our HDM constraint operators, and which allow us to transform between different modelling languages. In particular, the relational HDM model in Figure 3(b) may be transformed into the ER HDM model in Figure 2(b) by applying a sequence of transformations using the equivalence relationships presented in the following four subsections. Since the ORM HDM model in Figure 5(b) is the same as the ER HDM model in Figure 2(b), except for trivial renaming of constructs, then if we apply rename transformations to make the ORM HDM model match those in the ER HDM model, the same sequence of transformations may describe the mapping from relational to ORM models. Section 3.5 describes the fifth general purpose equivalent rule, and shows how it is used as part of the transformation of the UML HDM model in Figure 4(b) to the ER HDM model. However, the UML to ER transformation as a whole will be demonstrated to be non-equivalence preserving.

primitive transformation	reverse transformation
addNode($\langle\langle N\rangle\rangle$,$q$)	deleteNode($\langle\langle N\rangle\rangle$,$q$)
addEdge($\langle\langle E, N_1, \ldots, N_m\rangle\rangle$,$q$)	deleteEdge($\langle\langle E, N_1, \ldots, N_m\rangle\rangle$,$q$)
addConstraint($\langle\langle NE\rangle\rangle$ op $\langle\langle NE\rangle\rangle$)	deleteConstraint($\langle\langle NE_1\rangle\rangle$ op $\langle\langle NE_2\rangle\rangle$)
renameNode($\langle\langle N_1\rangle\rangle$,$\langle\langle N_2\rangle\rangle$)	renameNode($\langle\langle N_2\rangle\rangle$,$\langle\langle N_1\rangle\rangle$)
renameEdge($\langle\langle E_1\rangle\rangle$,$\langle\langle E_2\rangle\rangle$)	renameEdge($\langle\langle E_2\rangle\rangle$,$\langle\langle E_1\rangle\rangle$)
extendNode($\langle\langle N\rangle\rangle$)	contractNode($\langle\langle N\rangle\rangle$)
extendEdge($\langle\langle E, N_1, \ldots, N_m\rangle\rangle$)	contractEdge($\langle\langle E, N_1, \ldots, N_m\rangle\rangle$)
extendConstraint($\langle\langle NE\rangle\rangle$ op $\langle\langle NE\rangle\rangle$)	contractConstraint($\langle\langle NE_1\rangle\rangle$ op $\langle\langle NE_2\rangle\rangle$)

Table 1: Primitive transformations on the HDM

In order to give our mappings a rigorous basis, we define them around a HDM transformation language [21, 15] which allows the specification of bidirectional mappings between equivalent data sources, and also the specification of where one data source has greater information capacity than another. Table 1 lists those primitive transformations of the language that we use in this paper, and we now briefly review their semantics.

The first three rows of the table list add transformations, which have the semantics that the construct they add to the model is fully derivable from the existing model. In the case of nodes and edges, this means there is a query q supplied that gives the extent of the node or edge derived from the extent of nodes and edges already in the model. Use of these transformations therefore preserves information; there is no new information introduced by their use, the reverse delete transformations do not loose information from the model.

The next two rows of Table 1 describe how nodes or edges may be renamed, whilst preserving all their associations, and preserving their extent. The final three rows list extend transformations, which have the semantics that what they add to the model is not derivable from the existing model. Use of these transformations do not preserve information; and what they introduce to the new model is specific to that model.

3.1 Inclusion Merge

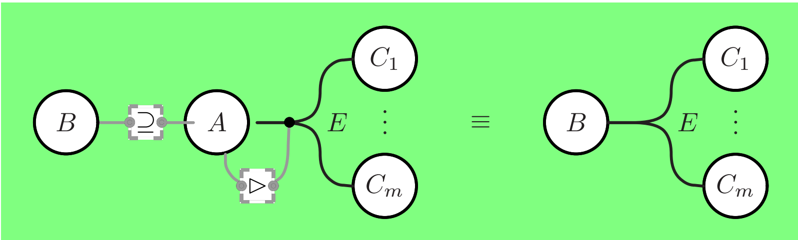


Figure 6: Equivalence Relationships: Inclusion Merge

The **Inclusion Merge** equivalence in Figure 6 allows us to merge the two nodes $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ together because $\langle\langle A\rangle\rangle$ is a subset of $\langle\langle B\rangle\rangle$ and there is a mandatory constraint from $\langle\langle A\rangle\rangle$ to an edge $\langle\langle E, A, \vec{C}\rangle\rangle$. The mandatory constraint is dropped as we merge $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ and the edge $\langle\langle E, A, \vec{C}\rangle\rangle$ now identifies the elements of $\langle\langle B\rangle\rangle$ that were in $\langle\langle A\rangle\rangle$. Any edges or constraints that applied to $\langle\langle B\rangle\rangle$ remain, and any other edges on $\langle\langle A\rangle\rangle$ are redirected to $\langle\langle B\rangle\rangle$. Definition 3.1 gives a pseudo code definition of this equivalence, that generates primitive transformations on the HDM. The pseudo code first iterates over all edges that link node $\langle\langle A\rangle\rangle$ with a set of nodes $\langle\langle \vec{C}\rangle\rangle$, and creates a new edge e' that links $\langle\langle \vec{C}\rangle\rangle$ with $\langle\langle B\rangle\rangle$ instead of $\langle\langle A\rangle\rangle$. For each of these new edges, all the constraints of the old edge e are copied across by the move_dependents function (defined in Definition 3.2), except before calling that function we delete all the mandatory constraints from $\langle\langle A\rangle\rangle$ to e to prevent them being copied. The final two lines of Definition 3.1 delete the constraint between $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$, and then deletes $\langle\langle A\rangle\rangle$, giving a list comprehension that can restore the values of $\langle\langle A\rangle\rangle$ from the non-mandatory edge $\langle\langle E, B, \vec{C}\rangle\rangle$.

Definition 3.1 Inclusion Merge

inclusion_merge($\langle\langle B\rangle\rangle$,$\langle\langle E, A, \vec{C}\rangle\rangle$)
  foreach $e \in Edges$ forwhich $e = \langle\langle E_a, A, \vec{N}\rangle\rangle$
    deleteConstraint($\langle\langle A\rangle\rangle \triangleright e$)
  endforeach;
  move_dependents($\langle\langle A\rangle\rangle$,$\langle\langle B\rangle\rangle$,id $\langle\langle A\rangle\rangle$);
  deleteConstraint($\langle\langle A\rangle\rangle \subseteq \langle\langle B\rangle\rangle$);
  deleteNode($\langle\langle A\rangle\rangle$,$[\{x\} \mid \{x, y\} <\!- \langle\langle E, B, \vec{C}\rangle\rangle]$);

□

The last line of the inclusion_merge makes use of the **intermediate query language** (**IQL**) [12, 13] of the AutoMed system, which is an implementation of list comprehensions [4]. The query produces a list of single arity tuples $\{x\}$ from a generator $\{x, y\} <\!- \langle\langle E, B, \vec{C}\rangle\rangle$ that iterates over the members of the list of binary tuples in the scheme $\langle\langle E, B, \vec{C}\rangle\rangle$.

The definition of move_dependents takes three arguments, the first two (a, b) of which must be a node or edge, and the third a mapping list that maps instances of a to instances of b.

For use in inclusion merge, the third argument should be an identity function id, definited as $id(\langle\langle A\rangle\rangle) = [\{a,a\} \mid a <\!\!- \langle\langle A\rangle\rangle]$.

Definition 3.2 Move Dependents

move_dependents(a,b,map)
    foreach $(a\ op\ d) \in Cons$ forwhich $b \neq d$
        addConstraint($b\ op\ d$); deleteConstraint($a\ op\ d$) endif
    endforeach;
    foreach $(d\ op\ a) \in Cons$ forwhich $b \neq d$
        addConstraint($d\ op\ b$); deleteConstraint($d\ op\ a$) endif
    endforeach;
    foreach $e \in Edges$ forwhich $e = \langle\langle F, a, \vec{N}\rangle\rangle$
        let e'=$\langle\langle F, b, \vec{N}\rangle\rangle$;
        addEdge(e',$[\{b,c\} \mid \{a,c\} <\!\!- e; \{a,b\} <\!\!- map]$);
        move_dependents(e,e', $[\{\{a,c\},\{b,c\}\} \mid \{a,b\} <\!\!- map; \{a,c\} <\!\!- e]$);
        deleteEdge(e,$[\{a,c\} \mid \{b,c\} <\!\!- e'; \{a,b\} <\!\!- map]$);
    endforeach;

□

In Example 3.1, the series of transformations that will convert the HDM model in Figure 3(b) into that in Figure 2(b) are listed. The first two steps in the series are applications of inclusion merge, which after 2 result in the intermediate HDM model shown in Figure 7.

Example 3.1 Transforming between relational and ER HDM models

1 inclusion_merge(⟨⟨student:name⟩⟩, ⟨⟨_,result:name,result⟩⟩)
2 inclusion_merge(⟨⟨course:code⟩⟩, ⟨⟨_,result:code,result⟩⟩)
3 identity_node_merge(⟨⟨_,ug:name,ug⟩⟩)
4 unique_mandatory_redirection(⟨⟨_,student:name,result⟩⟩, ⟨⟨_,student:name,student⟩⟩)
5 unique_mandatory_redirection(⟨⟨_,course:text,result⟩⟩, ⟨⟨_,course,result⟩⟩)
6 inclusion_edge_merge(⟨⟨_.result,student⟩⟩, ⟨⟨_.result,course⟩⟩)
7 move_dependents(⟨⟨_,student,student:name⟩⟩, ⟨⟨student⟩⟩, ⟨⟨_,student,student:name⟩⟩)

□

Taking transformation step 1 and applying Definition 3.1, we may expand the steps into a series of primitive transformation steps shown in Example 3.2. Step 1.1 is a result of the first foreach loop in Definition 3.1, Steps 1.2 and 1.3 result from the call to move_dependents, and 1.4 and 1.5 result from the last two lines of Definition 3.1.

Example 3.2 Primitive steps associated with transformation 1

1.1 deleteConstraint(⟨⟨result:name⟩⟩ $\rhd$ ⟨⟨_,result:name,result⟩⟩)
1.2 addEdge(⟨⟨_,student:name,result⟩⟩,
        [{b, c} | {a, c} <− ⟨⟨_,result:name,result⟩⟩; {a, b} <− id⟨⟨result:name⟩⟩]
1.3 deleteEdge(⟨⟨_,result:name,result⟩⟩,
        [{a, c} | {b, c} <− ⟨⟨_,student:name,result⟩⟩; {a, b} <− id⟨⟨result:name⟩⟩]
1.4 deleteConstraint(⟨⟨result:name⟩⟩ $\subseteq$ ⟨⟨student:name⟩⟩)
1.5 deleteNode(⟨⟨result:name⟩⟩,
        [{b} | {b, c} <− ⟨⟨_,student:name,result⟩⟩]

□

The expansion of transformations 1 illustrates that inclusion merge is a data preserving transformation, since the edge ⟨⟨_,result,student:name⟩⟩ may be recovered by the query in 1.3 (which in turn uses the query of 1.5 to find the extent of ⟨⟨result:name⟩⟩), and the new edge ⟨⟨_,result,student:name⟩⟩ may be derived from existing data in 1.2. A very similar expansion into primitive steps may be performed for 2, with a similar argument about data preservation.

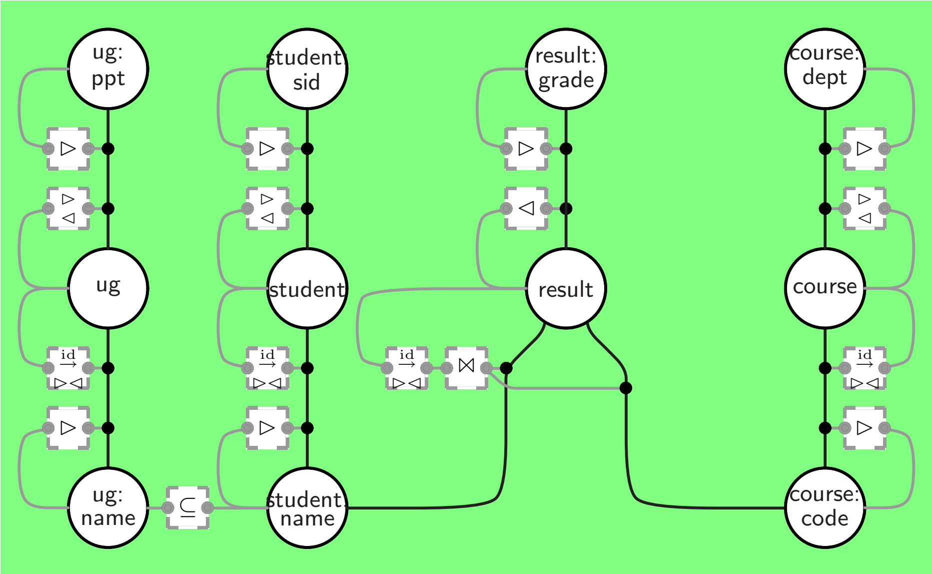


Figure 7: Intermediate HDM model in relational to ER conversion, after steps ① and ②

3.2 Identity Node Merge

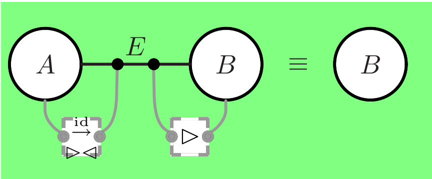
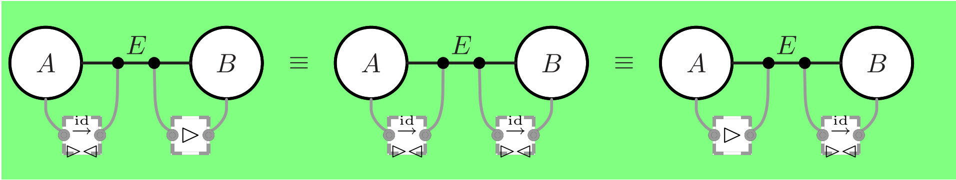


Figure 8: Equivalence Relationships: Identity Node Merge

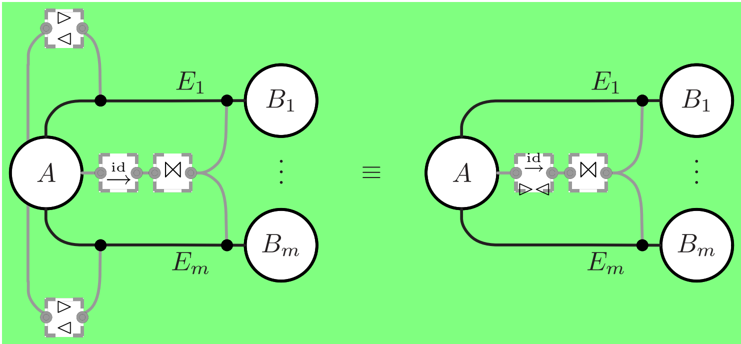
The **Identity Node Merge** in Figure 8 allows us to merge the two nodes $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ together because they are identical. The constraints $\langle\langle A\rangle\rangle \xrightarrow{\text{id}} \langle\langle E, A, B\rangle\rangle$, $\langle\langle A\rangle\rangle \rhd \langle\langle E, A, B\rangle\rangle$, and $\langle\langle A\rangle\rangle \lhd \langle\langle E, A, B\rangle\rangle$ taken together mean that every instance of the edge $\langle\langle E, A, B\rangle\rangle$ is an identity mapping for $\langle\langle A\rangle\rangle$, and there is exactly one such mapping in $\langle\langle E, A, B\rangle\rangle$ for every element of $\langle\langle A\rangle\rangle$. As each element in $\langle\langle E, A, B\rangle\rangle$ is an identity mapping, each element in $\langle\langle A\rangle\rangle$ must be in $\langle\langle B\rangle\rangle$. Conversely because we also have $\langle\langle B\rangle\rangle \rhd \langle\langle E, A, B\rangle\rangle$, each element in $\langle\langle B\rangle\rangle$ must be in $\langle\langle A\rangle\rangle$, and so $\langle\langle A\rangle\rangle = \langle\langle B\rangle\rangle$. This implies both $\langle\langle B\rangle\rangle \xrightarrow{\text{id}} \langle\langle E, A, B\rangle\rangle$ and $\langle\langle B\rangle\rangle \lhd \langle\langle E, A, B\rangle\rangle$, and thus the equivalences illustrated in Figure 9(a) hold. Because we have identified them as equal, nodes $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ can be merged together and the edge $\langle\langle E, A, B\rangle\rangle$ dropped, using Definition 3.3. Note any node can have this transformation applied in reverse, copying instances into a new node and linking the old node to the new node via an edge containing the identity instances.

Definition 3.3 Identity Node Merge

identity_node_merge($\langle\langle E, A, B\rangle\rangle$)
  let e=$\langle\langle E, A, B\rangle\rangle$;
  move_dependents($\langle\langle A\rangle\rangle$,$\langle\langle B\rangle\rangle$,$e$);
  foreach $c \in Cons$ forwhich contains(e,c)
    deleteConstraint(c);
  endforeach;



(a) Transposing identity constraint



(b) Mandatory-unique constraints in joins

Figure 9: Fundamental equivalences on HDM constraints

delEdge(e,[$\{x, x\}$ | $\{x\}$ <− $\langle\langle B\rangle\rangle$]);
deleteNode($\langle\langle A\rangle\rangle$,$\langle\langle B\rangle\rangle$);

$\square$

This identity mapping comes about by the way some modelling languages specify a certain attribute as being an entity's identifying attribute (such as the primary key constraint in the relational model).

In Figure 7 we can use identity node merge to merge nodes $\langle\langle$ug:name$\rangle\rangle$ and $\langle\langle$ug$\rangle\rangle$ by step 3 in Example 3.1. Note that the constraint $\langle\langle$ug:name$\rangle\rangle$ $\subseteq$ $\langle\langle$student:name$\rangle\rangle$ is not lost, but becomes $\langle\langle$ug$\rangle\rangle$ $\subseteq$ $\langle\langle$student:name$\rangle\rangle$. Figure 11 is partially derived by applying this merge.

3.3 Unique-Mandatory Redirection

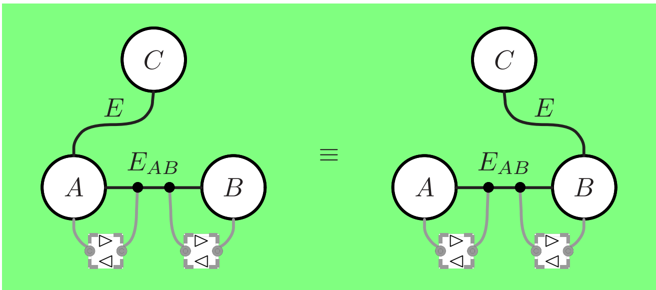


Figure 10: Equivalence Relationships: Unique-Mandatory Redirection

The **Unique-Mandatory Redirection** equivalence in Figure 10 allows us to move an edge $\langle\langle E, A, \vec{C}\rangle\rangle$ from node $\langle\langle A\rangle\rangle$ to node $\langle\langle B\rangle\rangle$ because both $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ have a unique and mandatory constraint on the common edge $\langle\langle E_{AB}, A, B\rangle\rangle$. These constraints together are equivalent to stating that there is a one to one correspondence between the elements of $\langle\langle A\rangle\rangle$ and $\langle\langle B\rangle\rangle$ so whatever is related to an element of $\langle\langle A\rangle\rangle$ through $\langle\langle E, A, \vec{C}\rangle\rangle$ is equally related to the corresponding element in $\langle\langle B\rangle\rangle$. Moving the edge requires us to rewrite the elements of the edge, replacing in each the value that came from $\langle\langle A\rangle\rangle$ with the corresponding value from $\langle\langle B\rangle\rangle$ (via $\langle\langle E_{AB}, A, B\rangle\rangle$).

Definition 3.4 Unique-Mandatory Redirection

unique_mandatory_redirection($\langle\langle E, A, \vec{C}\rangle\rangle$,$\langle\langle E_{AB}, A, B\rangle\rangle$)
let $e = \langle\langle E, A, \vec{C}\rangle\rangle$;
let $map = \langle\langle E_{AB}, A, B\rangle\rangle$;
if $(A \xrightarrow{\text{id}} e) \in Cons$ then exception endif;
let $e' = \langle\langle E, B, \vec{C}\rangle\rangle$;
addEdge(e',$[\{b, c\} \mid \{a, c\} <\!- e; \{a, b\} <\!- map]$);
move_dependents(e,e',map)
deleteEdge(e',$[\{a, c\} \mid \{b, c\} <\!- e'; \{a, b\} <\!- map]$);

□

For the HDM model in Figure 7, 4 in Example 3.1 moves the edge $\langle\langle$_,result,student:name$\rangle\rangle$ from node $\langle\langle$student:name$\rangle\rangle$ to node $\langle\langle$student$\rangle\rangle$, becoming edge $\langle\langle$_,result,student$\rangle\rangle$. This transformation does not loose information, because of the constraints on the edge $\langle\langle$_,student:name,student$\rangle\rangle$ (note that $\langle\langle$student:name$\rangle\rangle \lhd \langle\langle$_,student,student:name$\rangle\rangle$ is implied by the other constraints present on the edge; as illustrated in Figure 9(a)). Similarly we can apply 5 to move edge $\langle\langle$_,result,course:text$\rangle\rangle$ to become $\langle\langle$_,result,course$\rangle\rangle$ Applying these two edge redirections in addition to the previous identity node merge results in Figure 11.

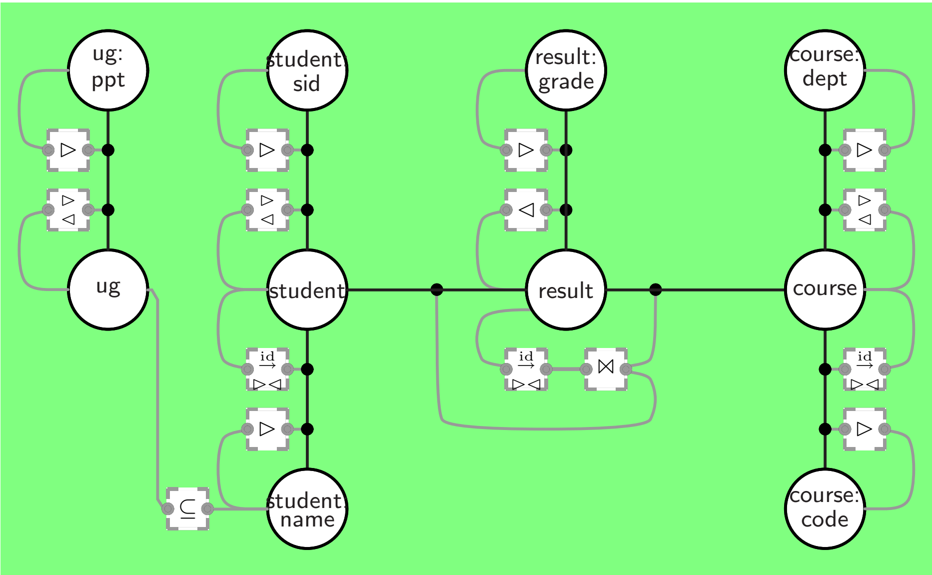


Figure 11: Intermediate HDM model in relational to ER conversion, after steps 1–5

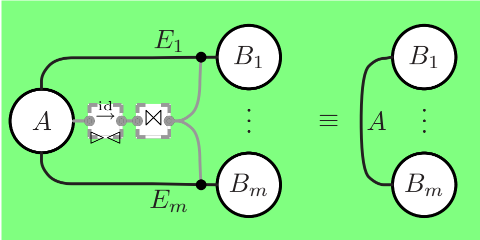


Figure 12: Equivalence Relationships: Identity Edge Merge

3.4 Identity Edge Merge

The **Identity Edge Merge** in Figure 12 allows us to replace a node $\langle\langle A\rangle\rangle$ with associated edges $\langle\langle E_1, A, B_1\rangle\rangle \ldots \langle\langle E_m, A, B_m\rangle\rangle$ with a single edge $\langle\langle A, B_1 \ldots B_m\rangle\rangle$. The constraints $\xrightarrow{\text{id}}$, $\triangleright$, and $\triangleleft$ between $\langle\langle A\rangle\rangle$ and the natural join of $\langle\langle E_1, A, B_1\rangle\rangle \ldots \langle\langle E_m, A, B_m\rangle\rangle$ mean that for each instance of node $\langle\langle A\rangle\rangle$ there is exactly one instance of the join of edges $\langle\langle E_1, A, B_1\rangle\rangle \ldots \langle\langle E_m, A, B_m\rangle\rangle$. We populate the new hyper edge $\langle\langle A, B_1 \ldots B_m\rangle\rangle$ using the corresponding values in $\langle\langle B_1\rangle\rangle \ldots \langle\langle B_m\rangle\rangle$ from each instance of the node $\langle\langle A\rangle\rangle$. Because of the identity mapping there is no information in the node $\langle\langle A\rangle\rangle$ that is not in this new edge.

Definition 3.5 Identity Edge Merge

inclusion_edge_merge($\langle\langle E_1, A, B_1\rangle\rangle, \ldots, \langle\langle E_m, A, B_m\rangle\rangle$)
    let $a = \langle\langle A, B_1, \ldots, B_m\rangle\rangle$;
    addEdge(a,
        $[\{b_1, \ldots, b_m\} \mid \{a, b_1\} <\!- \langle\langle E_1, A, B_1\rangle\rangle; \ldots; \{a, b_m\} <\!- \langle\langle E_m, A, B_m\rangle\rangle]$);
    foreach $(A\ op\ e) \in Cons$ forwhich $e \in \{\langle\langle E_1, A, B_1\rangle\rangle, \ldots, \langle\langle E_m, A, B_m\rangle\rangle\}$
        deleteConstraint($A\ op\ e$)
    endforeach;
    move_dependents($\langle\langle A\rangle\rangle$,$a$,id $\langle\langle$A$\rangle\rangle$);
    deleteNode($\langle\langle A\rangle\rangle$,$a$)

□

In Figure 11 we can use identity node merge to replace the node ⟨⟨result⟩⟩ with the edge ⟨⟨result,student,course⟩⟩, in step 6 of Example 3.1. In this case the new edge is binary because the natural join was between two edges. Note that as part of this process, the edge ⟨⟨_,result,result:grade⟩⟩ from ⟨⟨result⟩⟩ to ⟨⟨result:grade⟩⟩ becomes ⟨⟨_,⟨⟨result,student,course⟩⟩,result:grade⟩⟩.

All that is left for us to do in order to obtain the HDM ER model is to move ⟨⟨ug⟩⟩ $\subseteq$ ⟨⟨student:name⟩⟩ to ⟨⟨ug⟩⟩ $\subseteq$ ⟨⟨student⟩⟩. This is correct to do for similar reasons to the unique-mandatory redirection being correct for edges, but here we are moving a constraint between two nodes for which, in addition, we know the extent to be identical. This redirection is achieved by using the move dependents subroutine in 7, and the result is Figure 2(b).

3.5 Node Reidentify

Object orientation introduces the concept of there being an unique **object identifier** (**OID**) that is associated to instances of a class, and that OID is not represented as an attribute. Thus when we look at the HDM representation of the UML shown in Figure 4, although similar to those for the relational, ER and ORM models, there is no use of the $\xrightarrow{\text{id}}$ constraint made between nodes representing the UML class, such as ⟨⟨student⟩⟩, and ⟨⟨_,student,student:name⟩⟩. This is because ⟨⟨student⟩⟩ has as its extent the object identifiers of the student UML class, whilst ⟨⟨student:name⟩⟩ has as its extent the names of students.

Definition 3.6 Node Reidentify

node_reidentify($\langle\langle A\rangle\rangle$,$map$)
    addNode($\langle\langle A'\rangle\rangle$,$[\{b\} \mid \{a\} <\!- \langle\langle A\rangle\rangle; \{a,b\} <\!- map]$);
    foreach ($\langle\langle A_s\rangle\rangle \subseteq \langle\langle A\rangle\rangle) \in Cons$
        node_reidentify($\langle\langle A_s\rangle\rangle$,$map$)
    endforeach
    move_dependents($\langle\langle A\rangle\rangle$,$\langle\langle A'\rangle\rangle$,$map$);
    deleteNode($\langle\langle A\rangle\rangle$,$[\{a\} \mid \{b\} <\!- \langle\langle A'\rangle\rangle; \{a,b\} <\!- map]$)
    renameNode($\langle\langle A'\rangle\rangle$,$\langle\langle A\rangle\rangle$)

□

Example 3.3 Transforming between UML and ER HDM models
8 extendConstraint(⟨⟨student:name⟩⟩ ◁ ⟨⟨_,student,student:name⟩⟩)
9 inverse_identity_node_merge(⟨⟨student⟩⟩, ⟨⟨student:oid⟩⟩)
10 node_reidentify(⟨⟨student⟩⟩, [{x, y} |
    {o, x} <− ⟨⟨_,student,student:oid⟩⟩; {o, y} <− ⟨⟨_,student,student:name⟩⟩])
11 extendConstraint(⟨⟨course:code⟩⟩ ◁ ⟨⟨_,course,course:code⟩⟩)
12 inverse_identity_node_merge(⟨⟨course⟩⟩, ⟨⟨course:oid⟩⟩)
13 node_reidentify(⟨⟨course⟩⟩, [{x, y} |
    {o, x} <− ⟨⟨_,course,course:oid⟩⟩; {o, y} <− ⟨⟨_,course,course:code⟩⟩])
14 renameEdge(⟨⟨:has:exam,student,course⟩⟩, ⟨⟨result,student,course⟩⟩)

□

When transforming between between an OO model such as UML, and key based models such as ORM, ER or relational, we must overcome the fundamental difference in data modelling based on OIDs and natural keys. This will require us finding attributes or associations of the UML class that can be used to identify instances of the UML class.

Comparing the UML model in Figure 4 with the ER model in Figure 2, the HDM models of the two appear similar. One difference is trivial, in that the edge between ⟨⟨student⟩⟩ and ⟨⟨course⟩⟩ has a different name in the two models. The other difference is between the use of OIDs and natural keys, ER HDM model, using natural keys, has ⟨⟨student⟩⟩ $\xrightarrow{\text{id}}$ ⟨⟨_,student,student:name⟩⟩ and ⟨⟨course⟩⟩ $\xrightarrow{\text{id}}$ ⟨⟨_,course,course:code⟩⟩, whereas the UML HDM does not have these constraints. Example 3.3 lists a sequence of transformations that converts the UML model into an 'ER compatible' HDM model that has explicit attributes for the OIDs, and uses a natural key to identify the ER entity instances. The following steps explain the example:

1. Missing from the UML model is any definition of natural keys for the UML classes. Hence step 8 introduces a new constraint that indicates that name is a candidate key for student.

2. The inverse of identity node merge in step 9 generates a new node ⟨⟨student:oid⟩⟩, connected to ⟨⟨student⟩⟩ by a new edge ⟨⟨_,student,student:oid⟩⟩. If for example the node ⟨⟨student⟩⟩ had the extent [{&1}, {&2}, {&3}, {&4}] before this step, then after this step this edge will have as its extent [{&1, &1}, {&2, &2}, {&3, &3}, {&4, &4}].

3. Step 10 then has the net effect of repopulating the ⟨⟨student⟩⟩ node with values of the ⟨⟨student:name⟩⟩ attribute. If, before this step ⟨⟨_,student,student:name⟩⟩ had as its extent [{&1, 'Mary'}, {&2, 'John'}, {&3, 'Jane'}, {&4, 'Fred'}], then the *map* generated would be the same list, and hence after 10, ⟨⟨_,student,student:oid⟩⟩ would have the same list of values as its extent. The result of this step is shown in Figure 13

4. Steps 11–13 perform a similar conversion to the ⟨⟨course⟩⟩ node into a natural key based construct.

5. Step 14 deals with the trivial problem of renaming the edge between ⟨⟨student⟩⟩ and ⟨⟨course⟩⟩ to match the name in the ER model.

Note that at the end of this process, the ⟨⟨student:oid⟩⟩ and ⟨⟨course:oid⟩⟩ nodes can be discarded using contract transformations, to symbolise that the ER model looses the OID values of the UML model, or the ER model could be enhanced with oid attributes, if it was intended to use the ER model to fully represent the UML model.

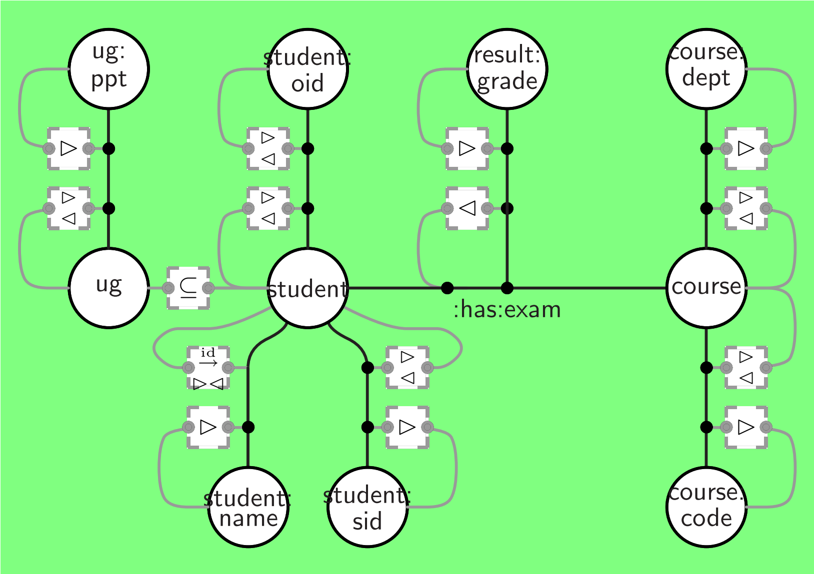


Figure 13: UML to ER mapping after 10

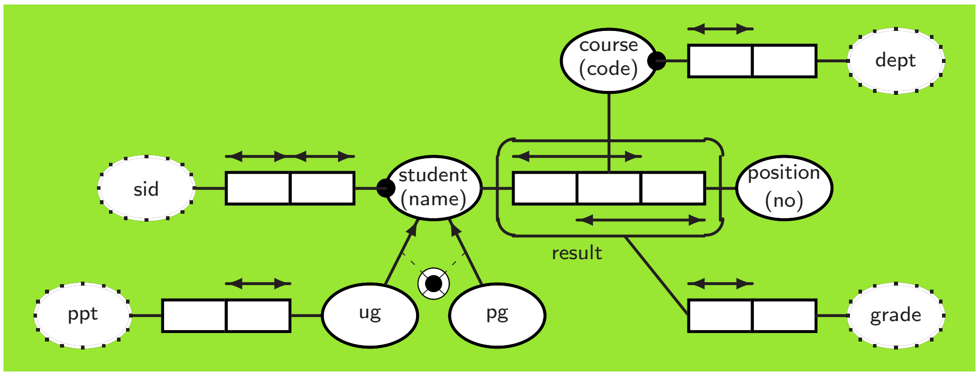
3.6 Non-Equivalent Models

The examples in Figures 2–5 were deliberately chosen to illustrate how we could draw an equivalence between models with the same information capacity. In practice, modelling languages have different expressive powers, and hence there may be no equivalent model.

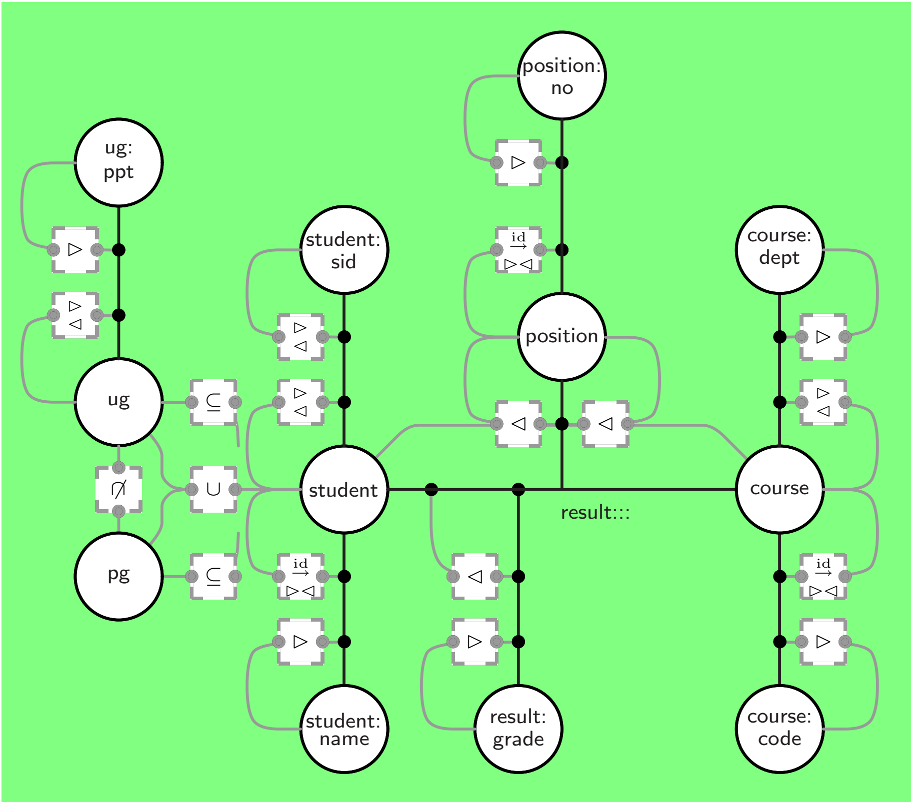
For example, changing the cardinality constraint in Figure 2(a) of student being associated with result from 0:N to 1:N would result in a ▷ being added between ⟨⟨student⟩⟩ and ⟨⟨result,student,course⟩⟩ in Figure 2(b). If we were to reverse the process outlined in section 4 with this extra constraint in place then we would run into a problem. The reversed edge redirection from ⟨⟨_,result,student⟩⟩ in Figure 11 to ⟨⟨_,result,student:name⟩⟩ in Figure 7 carries the mandatory constraint introduced by 1:N. When we come to reverse the inclusion merge that merged ⟨⟨result:name⟩⟩ ⊆ ⟨⟨student:name⟩⟩ to enable the relationship between ⟨⟨result⟩⟩ and ⟨⟨student:name⟩⟩ to be represented as a foreign key we loose ⟨⟨student:name⟩⟩ ▷ ⟨⟨result⟩⟩. This is precisely because the relational schema in Figure 3(a) does not express the fact that every student.name must be referenced by at least one result.name. This lost constraint is, therefore, not a weakness in the approach, but an example of the approach formally identifying what information from the ER schema cannot be represented in the relational model. In this particular case, it might appear that we could repair the relational model by the addition of the foreign key constraint student.name → result.name, but this would not be legal since result.name is not a candidate key of result.

4 Handling Additional Modelling Concepts

Figure 14(a) illustrates an ORM model of an extended version of the student-course database where, as we will show, the ORM model is able to represent some aspect of the UoD that one or more of our other data models is unable to represent. The additions made to the ORM model of Figure 5 are as follows:



(a) ORM model of extend student-course database



(b) HDM representation of the ORM model

Figure 14: An ORM model of an extended student-course database

- The reference/predicate between value ⟨⟨sid⟩⟩ and entity ⟨⟨student⟩⟩ now has both roles as being key.

  This concept can be represented in a relational model with the ⟨⟨student,sid⟩⟩ attribute being a candidate key. Neither the ER nor UML models have a method of representing this concept however.

Typically, ER models do not make explicit the relationships between an entity and its attributes, but instead use some sort of syntax with an attribute's name to indicate that it is (or is part of) the primary identifier: no other uniqueness constraints can be expressed. With the more elaborate ER syntax where attribute/entity cardinality constraints are explicit, a one-to-one relationship is synonymous with the primary identifier and therefore can only be expressed once per entity.

Because of UML's reliance on object identifiers, it does not require classes to have value-based reference schemes and indeed requires nonstandard extensions to its notation to express an attribute's uniqueness in its association with its class.

Note that in our HDM production rules for UML there is no rule that generates a uniqueness constraint from an attribute to the edge associating it to its class. In our HDM production rules for ER the only way to generate this constraint is in conjunction with a reflexive constraint.

If we were to convert our extended ORM model into an HDM model that could have been produced by an equivalent ER schema, we would have to drop the uniqueness constraint from either ⟪sid⟫ or ⟪student⟫, or extend the ER language to handle candidate keys. For UML, both uniqueness constraints must be dropped, since it has no support for any keys.

- There is an additional subclass entity ⟪pg⟫ that is disjoint from ⟪ug⟫, and in addition, ⟪pg⟫ and ⟪ug⟫ are total w.r.t. entity ⟪student⟫.

  The disjointness and totality can not be represented in the relational model, since the relational model has no constructs that make use of either the HDM disjoint or union constraints. If we wanted to model our extended ORM example using a relational schema we would have to drop the exclusion constraint between ⟪ug⟫ and ⟪pg⟫ as well as the union constraint between these subsets and ⟪student⟫.

  There are several ways we may attempt to model the total partition.

  - We could represent each subset with a table containing the attributes common to just that subset, and a primary key that is also a foreign key to the supertype ⟪student⟫. The problem is that there is no way to enforce that at least (or at most) one instance of some referring foreign key exists for each instance of a primary key.
  - With some more elaborate transformations we could change the subclass identification into an attribute of the superclass telling us which subclass table we should join each instance to. This would enforce the exclusion constraint, and if we made the attribute mandatory it would also enforce the union constraint. The problem is that the relational model has no way of specifying this dynamic join constraint.
  - With yet more transformations we could use the subclass-identifying attribute as above and make each attribute of each subtype an optional attribute of the supertype where it is set to null when not applicable. Again, the relational model has no way of specifying that certain attributes must, or must not, be null depending on another attribute.

  Despite there being several ways to model subclasses in the relational model we can not retain exclusion and union constraints involving ⟪ug⟫ and ⟪pg⟫.

- The fact/predicate between entities ⟪student⟫, ⟪course⟫, and ⟪position⟫ has two overlapping keys, which states that any pair of ⟪student⟫ and ⟪course⟫ instances may appear at most once in the fact, and also that any pair of ⟪position⟫ and ⟪course⟫ instances may appear at most once in the fact.

  This concept of overlapping keys is not representable in the ER modelling language as we defined it in Section 2.1, since we choose to use cardinalities to denote number of occurrences of a single entity in the relationship. The UML model is capable of describing this concept, since it uses multiplicity to denote the number of occurrences of $n-1$ classes in an n-ary association.

5 Future Work

Currently we are compiling the minimal list of atomic constraints needed to represent all the constructs in all the common modelling languages, and the minimal set of equivalence rules that will allow us to manually convert a schema from one modelling language to another. The work reported to date results from our examination of relational, UML, ER and ORM models, and this will be extended to cover YAT [5] (semi-structured), XML Schema data models, and those features of ORM and UML not covered in this paper.

Once this work is complete, we will develop an algorithm to automate the conversion process, by searching for constructs in the target modelling language that can be constructed from the source schema's HDM in such a way as to minimise the number of constraints left in the HDM that have no corresponding target language construct. When there are no constraints left, the resulting target schema should be equivalent to the source schema. Otherwise semantic information is lost in the conversion, and the unmatchable constraints will tell us precisely what information has been lost.

We believe that the framework we have presented in this paper will be of use in formally comparing modelling languages and their expressibility, and that the proposed algorithm development will be of use in data integration.

References

[1] M. Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. In *Proc. ER'94*, LNCS, pages 403–419. Springer, 1994.

[2] 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.

[3] 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.

[4] P. Buneman et al. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.

[5] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! *SIGMOD Record*, 27(2):177–188, 1998.

[6] C.J. Date. Object indentifiers vs. relational keys. In *Relational Database: Selected Writings 1994–1997* [9].

[7] C.J. Date. *An Introduction to Database Systems*. Addison-Wesley, 8th edition edition, 2004.

[8] C.J. Date, H. Darwen, and N.A. Lorentzos. *Tempora Data and the Relational Model*. 2003, 2003.

[9] C.J. Date, H. Darwen, and D. McGoveran. *Relational Database: Selected Writings 1994–1997*. Addison-Wesley, 1998.

[10] P. Hall, J. Owlett, and S.J.P. Todd. Relations and entities. In G.M. Nijssen, editor, *Modelling in Data Base Management Systems*. North-Holland, 1975.

[11] T. Halpin. *Information Modeling and Relational Databases*. Academic Press, 2001.

[12] E. Jasper, A. Poulovassilis, and L. Zamboulis. Processing IQL queries and migrating data in the AutoMed toolkit. Technical Report No. 20, AutoMed, 2003.

[13] 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.

[14] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.

[15] 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.

[16] P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.

[17] R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Information Systems*, 19(1):3–31, 1994.

[18] S. Patig. Measuring expressiveness in conceptual modeling. In *Proc. CAiSE2004*, volume 3084 of *LNCS*, pages 127–141. Springer-Verlag, 2004.

[19] J-M. Petit, F. Toumani, J-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of denormalized relational databases. In *Proc. ICDE'96*, pages 218–227, 1996.

[20] 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.

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

[22] K. Schewe. Design theory for advanced datamodels. In *Proc. 12th Australasian Conf. on Database Technologies*, pages 3–9, 2001.

[23] R. Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, 1998.

[24] C. Zaniolo and M. Melkanoff. A formal approach to the definition and the design of conceptual schemata for database systems. *ACM TODS*, 1982.