Towards a Semi-Automated Approach to Intermodel Transformation

Michael Boyd[1] and Peter McBrien[2]

[1] PSA Parts Ltd, London SW19 3UA, mb@psaparts.co.uk
[2] Dept. of Computing, Imperial College, London SW7 2AZ, pjm@doc.ic.ac.uk

Abstract. This paper introduces an extension to the hypergraph data model used in the AutoMed data intergration approach that allows constraints common in static data modelling languages to be represented by a small set of primitive constraint operators. A set of equivalence rules are defined for this set of primitive constraint operators, and demonstrated to allow a mapping between relational, ER or UML models to be defined. The approach provides both a precise framework in which to compare data modelling languages, and forms the platform for further work in automating the process of transforming between different data modelling languages.

1 Introduction

The AutoMed project has developed the first implementation [2, 7] of a data integration technique called **both-as-view** (**BAV**) [10], which subsumes the expressive power of other published data integration techniques **global-as-view** (**GAV**), **local-as-view** (**LAV**), and **global-local-as-view** (**GLAV**) [8].

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 [9], 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 model, relational model or UML, to the constructs in a single lower level common data modelling language called the **hypergraph data model** (**HDM**) [14]. Figure 1 illustrates this concept being applied to three higher level models, all being related to the same underlying HDM graph.

In [9] 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 and UML higher level modelling languages – perform **intermodel transformations** – by the application of four types of equivalence rules on the HDM and its primitive constraint operators. 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

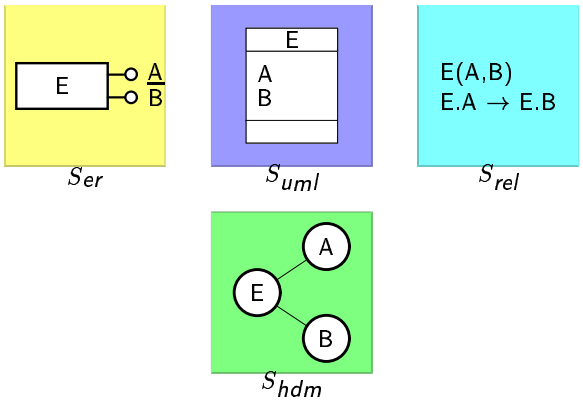


Fig. 1. Multiple models based on the HDM

ER conversion [1, 12], ORM to UML and relational conversion [6], and relational to generic object oriented conversion [4]. 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 to conversion between ER, relational and UML static class 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.

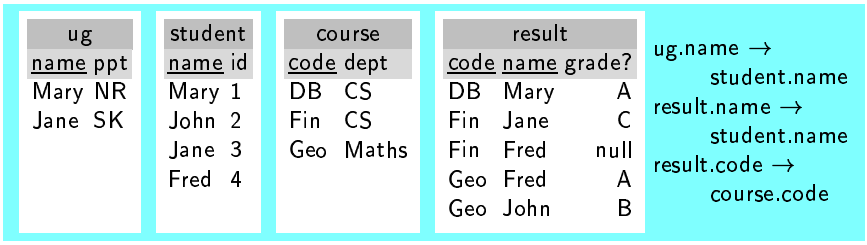
The paper is structured as follows. Section 2 reviews how AutoMed describes higher level data modelling languages by relating them to a graph structure. Then in Section 3 we propose a set of HDM constraint operators that form a language used to model the constraints in higher level data modelling languages. Section 4 details how we approach the transformation between modelling languages by applying equivalence rules to the graph.

2 Describing a Data Modelling Language

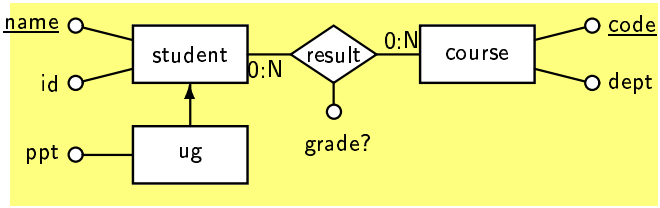
In [9] a general technique was proposed for the modelling of any structured data modelling language in the HDM. The premise of this approach is that in any data modelling language, the various constructs of the language can be viewed as being a combination of sets of values, and constraints between these sets, in a graph based model. This concept has been used in modelling relational models [17], and for OO and ER models [16], and is an intuitive assumption to make. 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

hyper-edges[1], and *Cons* is a set of constraint expressions over the *Nodes* and *Edges*. 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 [14]. The HDM is a simplication of the hypergraph model in [13], which allowed nodes to contain complete graphs.

It should be also noted that [16] argues for simplicity in modelling languages, and so using the simple the 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 using 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.



(a) Relational database schema and data

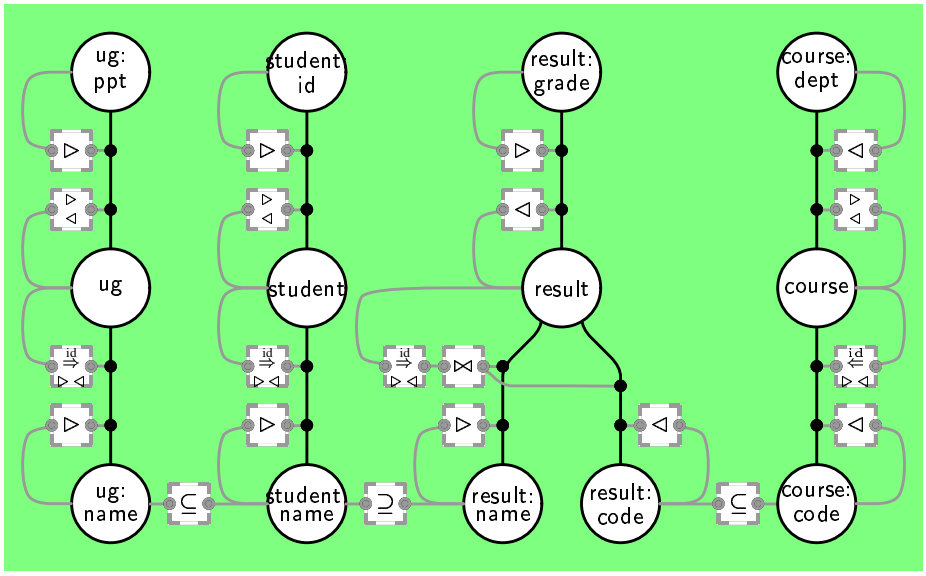


(b) ER model used to design relational database

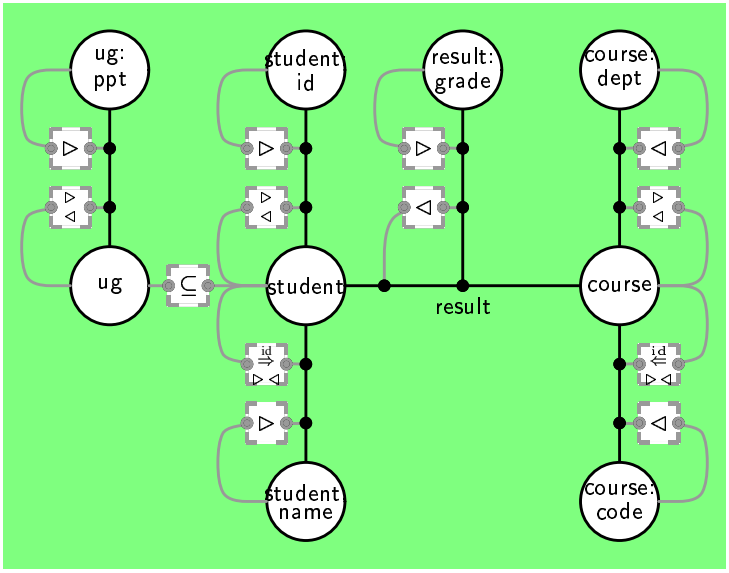
Fig. 2. Example schemas

Figure 2 shows two data models, which are equivalent in the sense that they have the same information capacity [11], yet are structured in different ways and in different modelling languages. The schemas represent a record of students and the courses that they sit, and the grade 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 both models indicate what are key attributes, and a ques-

[1] A hyper-edge 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.



(a) HDM representation of relational database schema



(b) HDM representation of the ER model

Fig. 3. HDM representation of Example schemas

tion mark follows a nullable attribute. In the relational model, foreign keys are shown by using a implication between the foreign key columns and the referenced table columns. In the ER model this foreign key may either be represented as 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).

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:

- A **nodal** construct is one that may appear in isolation in a model, such as a relational table or 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 relational table student by the scheme ⟨⟨student⟩⟩, and the table result by ⟨⟨result⟩⟩, and the ER entity student by ⟨⟨student⟩⟩.
  Figure 3 gives the HDM graphs that is implied by the definition above for the data models given in Figure 2. In the HDM graphs, each relational model table and ER model entity is represented by a HDM node of the same name, where HDM nodes are shown as large white circles with a black outline.
- 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 it by the scheme made up of the of the name of the HDM edge used to represent the construct, together with constraint expressions for the cardinality constraints, for example representing the result relationship by the scheme ⟨⟨result, student, 0:N, course, 0:N⟩⟩. In the diagrams, HDM edges are shown as thick black lines. The representation of the constraint part of the scheme will be discussed in the next section.
- A **link-nodal** construct is one that has associated values, but may only exist when associated with some other construct, such as attributes in the relational and ER models. In AutoMed, we represent attributes in the relational model by a scheme containing a HDM node that represents the construct it depends on, followed by the name of the HDM node that it represents, followed by the constraint on whether the attribute may be null. For example, the name column of table student is represented by the scheme ⟨⟨student, name, notnull⟩⟩, as is the name attribute of the entity student. In the HDM, this becomes a node ⟨⟨student:name⟩⟩ to represent values of the column/attribute, and the nameless edge ⟨⟨_,student,student:name⟩⟩ to represent the association of these values to table/entity ⟨⟨student⟩⟩. The fact that column/attribute names are prefixed by the associated table/entity name reflects a choice made when defining the construct in AutoMed. The user 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.
- A **constraint** construct is one that has no associated extent, but instead limits the extent of the constructs it connects to. In AutoMed, we represent the foreign key relationship 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. For example, the foreign key between ug and student is represented by the scheme $\langle\langle \mathsf{ug\_fk}, \mathsf{ug}, \langle\langle \mathsf{ug}, \mathsf{name}\rangle\rangle, \mathsf{student}, \langle\langle \mathsf{student}, \mathsf{name}\rangle\rangle\rangle\rangle$.
The ER model subset relationship is modelled as scheme giving the superset entity followed by the subset entity. For example, the subset between ug and student is represented by the scheme $\langle\langle \mathsf{student}, \mathsf{ug}\rangle\rangle$.

3 Modelling Constraints

We now introduce to the HDM a set of five 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, and NE any node or edge.

- **Inclusion** $NE_1 \subseteq NE_2$: The extent of node or edge NE_1 is a subset of node or edge NE_2. For example, the ER subset scheme $\langle\langle \mathsf{student}, \mathsf{ug}\rangle\rangle$ is represented by the constraint $\langle\langle \mathsf{ug}\rangle\rangle \subseteq \langle\langle \mathsf{student}\rangle\rangle$. In Figure 3(b) this constraint is shown in a grey dashed box, that links the HDM nodes that represent the entities $\langle\langle \mathsf{ug}\rangle\rangle$ and $\langle\langle \mathsf{student}\rangle\rangle$. In a similar way, the relational foreign key scheme $\langle\langle \mathsf{ug\_fk}, \mathsf{ug}, \langle\langle \mathsf{ug}, \mathsf{name}\rangle\rangle, \mathsf{student}, \langle\langle \mathsf{student}, \mathsf{name}\rangle\rangle\rangle\rangle$ is represented by a subset on the node representing the name attribute $\langle\langle \mathsf{ug{:}name}\rangle\rangle \subseteq \langle\langle \mathsf{student{:}name}\rangle\rangle$.
- **Exclusion** $\not\emptyset(NE_1 \ldots NE_m)$: The extents of a set of nodes or edges must be exclusive. For example, it is often the case that the generalisation construct in an ER modelling language implies the existence of an exclusion constraint between the HDM nodes that represent the child entities of the generalisation. Our example schemas do not include a use of this constraint.
- **Mandatory** $NE \rhd E$: node or edge NE is connected by edge E, and every instance in the extent NE must appear at least once in the extent of E. For example, the relational notnull constraint on columns implies that the mandatory constraints $\langle\langle \mathsf{result}\rangle\rangle \rhd \langle\langle \_, \mathsf{result}, \mathsf{result{:}code}\rangle\rangle$ and $\langle\langle \mathsf{result}\rangle\rangle \rhd \langle\langle \_, \mathsf{result}, \mathsf{result{:}name}\rangle\rangle$ apply between the node that represents a table result and the edges that partially represents the columns.
  Note that there is no such constraint on grade since it is a nullable column. The mandatory constraint also applies to all nodes that partially represent the column, forcing them to be in the edge that completes the representation of the column. For example, on the columns on the result table have the constraints $\langle\langle \mathsf{result{:}code}\rangle\rangle \rhd \langle\langle \_, \mathsf{result}, \mathsf{result{:}code}\rangle\rangle$, $\langle\langle \mathsf{result{:}name}\rangle\rangle \rhd \langle\langle \_, \mathsf{result}, \mathsf{result{:}name}\rangle\rangle$ and $\langle\langle \mathsf{result{:}grade}\rangle\rangle \rhd \langle\langle \_, \mathsf{result}, \mathsf{result{:}grade}\rangle\rangle$.
  In Figure 3 the mandatory constraint is shown in the same type of graphical notation as an inclusion constraint, but it should be noted that $NE \rhd E = E \lhd NE$, and that the left and right nodes of the constraint box correspond to the left and right side of the enclosed operator. Therefore the $\langle\langle \mathsf{course{:}dept}\rangle\rangle$ to $\langle\langle \_, \mathsf{course}, \mathsf{course{:}dept}\rangle\rangle$ mandatory constraint is drawn using $\lhd$ in the constraint box.
- **Unique** $NE \lhd E$: The node or edge NE is connected by edge E, and no instance in the extent NE may appear more than once in the extent of E. For example, this

constraint holds for all relational columns (forcing them not to be multivalued). Thus the columns of result have $\langle\langle$result$\rangle\rangle \lhd \langle\langle$_,result,result:code$\rangle\rangle$, $\langle\langle$result$\rangle\rangle \lhd \langle\langle$_,result,result:name$\rangle\rangle$, and $\langle\langle$result$\rangle\rangle \lhd \langle\langle$_,result,result:grade$\rangle\rangle$. Again note that $NE \lhd E = E \rhd NE$.

– **Identity** $NE \stackrel{id}{\Rightarrow} NE_{id}$: If an instance of NE appears in NE_{id}, then one of those instances must be an identity tuple: *i.e.* if $\langle a_1, \ldots, a_m \rangle$ of NE appears in a tuple of NE_{id}, then one of those tuples is $\langle \langle a_1, \ldots, a_m \rangle, a_1, \ldots, a_m \rangle$. The primary key construct of the relational modal specifies a natural join between its key attributes and gives the extent we associate with identifying the table. For example, the primary key of $\langle\langle$result$\rangle\rangle$ would be represented in the HDM by $\langle\langle$result$\rangle\rangle \stackrel{id}{\Rightarrow} (\langle\langle$_,result,result:code$\rangle\rangle \bowtie \langle\langle$_,result,result:name$\rangle\rangle)$.

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:

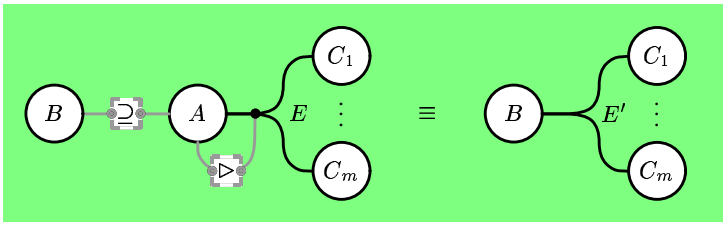
None $\rightarrow NE$ has 0:N occurrences in E
$NE \rhd E$ $\rightarrow NE$ has 1:N occurrences in E
$NE \lhd E$ $\rightarrow NE$ has 0:1 occurrences in E
$NE \rhd E \wedge NE \lhd E \rightarrow NE$ has 1:1 occurrences in E

Most of these constraint operators have been used before in the context on modelling single modelling languages. In particular, mandatory and unique constraints have been used in a hypergraph model for relational schemas in [17], and inclusion constraints appear in [15]. We believe that the identity constraint is new, and has the special property that is may be applied to state that the values involved in a mapping are identical to each other when they appear in the mapping. In combination with manadatory and unique, it allows for the description of keys in a data modelling language. For example, the relational model in Figure 3(a) uses $\rhd$, $\lhd$ and $\stackrel{id}{\Rightarrow}$ between node $\langle\langle$student$\rangle\rangle$ and edge $\langle\langle$_,student,student:name$\rangle\rangle$ to represent that $\langle\langle$student:name$\rangle\rangle$ is the key from $\langle\langle$student$\rangle\rangle$. By contrast, in HDM graph for a UML model in Figure 6(b) has no $\stackrel{id}{\Rightarrow}$ between $\langle\langle$student$\rangle\rangle$ and edge $\langle\langle$_,student,student:name$\rangle\rangle$ since there is no concept of key in UML.

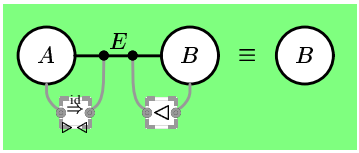
4 Inter Model Transformations

We now introduce four 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(a) may be transformed into the ER HDM model in Figure 3(b) by applying a sequence of transformations using the equivalence relationships in Figure 4.

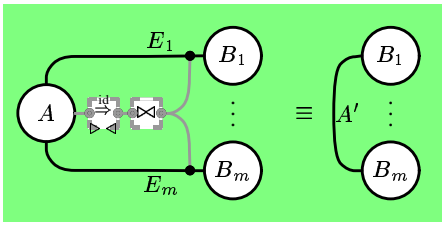
1. The **Inclusion Merge** equivalence in Figure 4 (a) allows us to merge the two nodes A and B together because A is a subset of B and there is a mandatory constraint from A to an edge E. The mandatory constraint is dropped as we merge A and B and the edge E now identifies the elements of B that were in A. Any edges or constraints that applied to B remain.



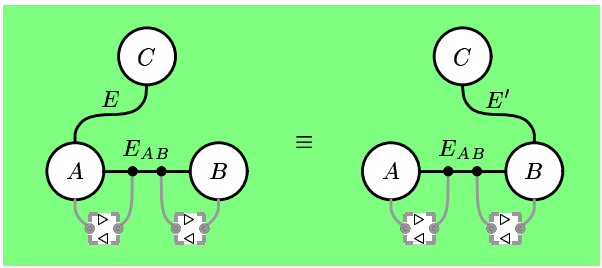
(a) Inclusion Merge



(b) Identity Node Merge



(c) Identity Edge Merge



(d) Unique-Mandatory Redirection

Fig. 4. Equivalence Relationships Between Constraints

Applying this to Figure 3(a) we see that node A=⟨⟨result:name⟩⟩ is a subset of B=⟨⟨student:name⟩⟩, and there is a mandatory constraint from ⟨⟨result:name⟩⟩ to the edge E=⟨⟨_,result,result:name⟩⟩, so we can merge the two nodes together. A similar argument allows us to merge ⟨⟨result:code⟩⟩ and ⟨⟨course:code⟩⟩ and the two applications result in Figure 5(a). These transformations do not loose information, since the edge ⟨⟨_,result,student:name⟩⟩ identifies the subset of ⟨⟨student:name⟩⟩ that was ⟨⟨result:name⟩⟩, and the edge ⟨⟨_,result,course:code⟩⟩ identifies the subset of ⟨⟨course:code⟩⟩ that was ⟨⟨result:code⟩⟩.

2. The **Identity Node Merge** in Figure 4(b) allows us to merge the two nodes A and B together because they are identical. $A \overset{\text{id}}{\Rightarrow} \triangleright \triangleleft E$ taken together mean that every instance of the edge E is an identity mapping for A, and there is exactly one such

mapping in E for every element of A. As each element in E is an identity mapping each element in A must be in B. Conversely because we also have $B \triangleright E$, each element in B must be in A, and so $A = B$ and $B \overset{\text{id}}{\Rightarrow} E$. Incidentally $B \triangleleft A$ is also implied. A and B can therefore be merged together and the edge E dropped.

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

In Figure 5(a) we can use identity node merge to merge nodes A=⟨⟨ug:name⟩⟩ and B=⟨⟨ug⟩⟩. Note that the constraint ⟨⟨ug:name⟩⟩ $\subseteq$ ⟨⟨student:name⟩⟩ is not lost, but becomes ⟨⟨ug⟩⟩ $\subseteq$ ⟨⟨student:name⟩⟩. Figure 5(b) is partially derived by applying this merge.

3. The **Unique-Mandatory Redirection** equivalence in Figure 4(d) allows us to move an edge E from node A to node B because both A and B have a unique and mandatory constraint on the common edge E_{AB}. These constraints together are equivalent to stating that there is a one to one correspondence between the elements of A and B so whatever is related to an element of A through E is equally related to the corresponding element in B. Moving the edge requires us to rewrite the elements of the edge, replacing in each the value that came from A with the corresponding value from B (via E_{AB}).

   In Figure 5(a) We can therefore move the edge E=⟨⟨_,result,student:name⟩⟩ from node A=⟨⟨student:name⟩⟩ to B=⟨⟨student⟩⟩, becoming edge E'=⟨⟨_,result,student⟩⟩, because of the constraints on the edge E_{AB}=⟨⟨_,student:name,student⟩⟩ (note that ⟨⟨student:name⟩⟩ $\triangleleft$ ⟨⟨_,student,student:name⟩⟩ is implied by the other constraints present on the edge). Similarly we may move edge ⟨⟨_,result,course:text⟩⟩ to become ⟨⟨_,result,course⟩⟩. Applying these two edge redirections in addition to the previous identity node merge in item (2) results in Figure 5(b).
4. The **Identity Edge Merge** in Figure 4(c) allows us to replace the node A and edges $E_1 \ldots E_m$ with the single edge A. The $\overset{\text{id}}{\Rightarrow} \triangleright \triangleleft$ between A and the natural join of $E_1 \ldots E_m$ mean that for each element of A there is exactly one element in each edge $E_1 \ldots E_m$. We populate the new hyper edge A between $B_1 \ldots B_m$ using the corresponding values in $B_1 \ldots B_m$ from each element of the node A. Because of the identity mapping there is no information in the node A that is not in this new edge.

   In Figure 5(b) we can use this to replace the node A=⟨⟨result⟩⟩ with the edge A'=⟨⟨result,student,course⟩⟩ in Figure 3(a). 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⟩⟩.
5. The last step requires us to move ⟨⟨ug⟩⟩ $\subseteq$ ⟨⟨student:name⟩⟩ to ⟨⟨ug⟩⟩ $\subseteq$ ⟨⟨student⟩⟩. Even though there is a unique and mandatory constraint on each end of the edge ⟨⟨_,student,student:name⟩⟩, we cannot simply redirect the constraint as we would do for an edge. To do that, we would require an identity redirection equivalence rule (which would be perfectly valid). We can, however, compose this identity redirection equivalence rule using two applications of the identity node merge.

   First we apply identity node merge to ⟨⟨student⟩⟩ and ⟨⟨student:name⟩⟩, merging them into ⟨⟨student⟩⟩. This in itself results in the subset constraint being rewritten

to conform with Figure 3(b). Now we just have ⟨⟨student:name⟩⟩ missing, which we recover by applying identity node merge in reverse. As this is an equivalence, there is nothing to stop us creating a new node ⟨⟨student:name⟩⟩ with the same extent as ⟨⟨student⟩⟩ and creating the edge ⟨⟨_,student,student:name⟩⟩ with the $\stackrel{id}{\Rightarrow} \rhd \lhd$ constraints. In doing so we choose to leave ⟨⟨ug⟩⟩ ⊆ ⟨⟨student⟩⟩ where it is, and the final result is the same HDM graph as in Figure 3(b), which is the HDM graph of the equivalent ER model.

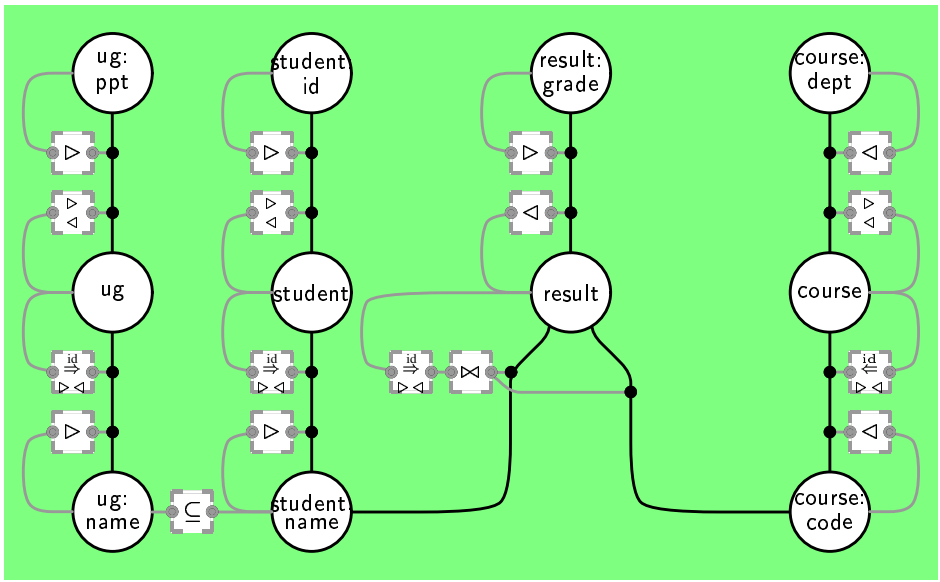
4.1 Non-Equivalent Models

The example in Figure 2 was 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(b) 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 3(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 5(b) to ⟨⟨_,result,student:name⟩⟩ in Figure 5(a) 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 2(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, we may repair the relational model by the additional of the foreign key constraint student.name → result.name.

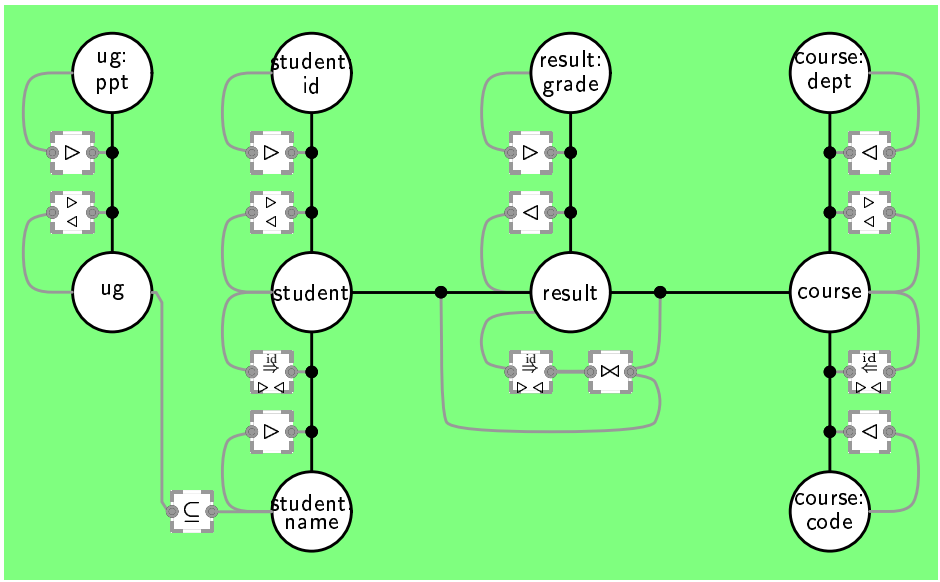
4.2 Equivalence between OO and non-OO models

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 UML shown in Figure 6, although similar to those for the earlier relational and ER models, there is no use of the $\stackrel{id}{\Rightarrow}$ 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.

When transforming between UML and ER or relational models, we must take attributes of the UML class, and use them to identify instances of the UML class. In our example, we may transform the HDM representation of the UML model into the HDM representation of the ER model by the following steps (which form a general template for OO to non-OO model conversion):

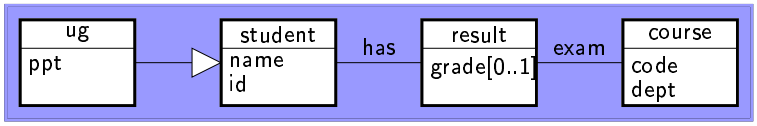


(a) After two applications of inclusion removal

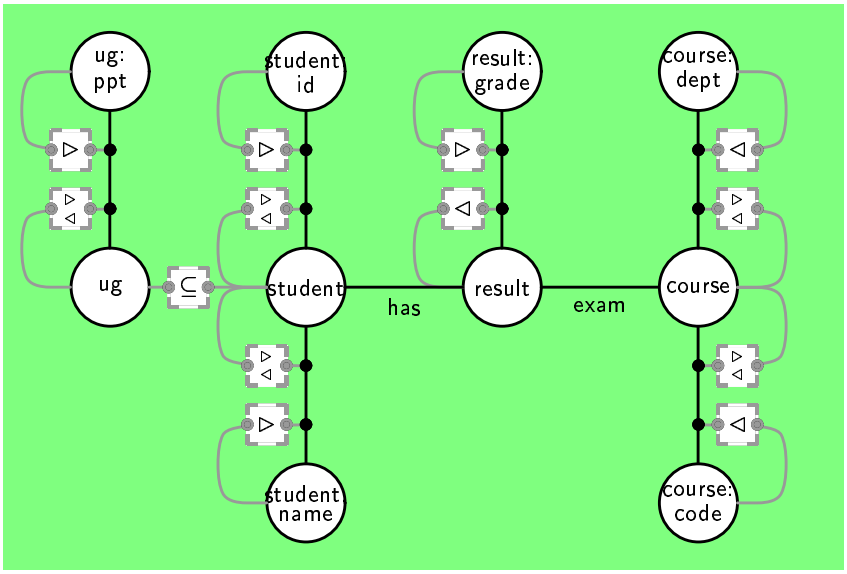


(b) After two applications of mandatory-unique redirection, and one identity node merge

Fig. 5. Example of application of equivalences



(a) UML class model for the student database



(b) HDM graph for UML model

Fig. 6. UML model for example schema

1. First introduce a unique constraint on the identifying attribute of a class. In the example, this would mean introducing a unique constraint from ⟨⟨student:name⟩⟩ to ⟨⟨_,student,student:name⟩⟩, to indicate that there is a one to one mapping between class ⟨⟨student⟩⟩ and attribute ⟨⟨student:name⟩⟩.
2. Use mandatory-unique direction to change the association of edges with the node representing the class to instead associate with the identifying attribute. In the example, we would redirect ⟨⟨has,student,result⟩⟩ from ⟨⟨student⟩⟩ to ⟨⟨student:name⟩⟩, making it become ⟨⟨has,student:name,result⟩⟩, and redirect ⟨⟨_,student,student:id⟩⟩ from ⟨⟨student⟩⟩ to ⟨⟨student:name⟩⟩, making it become ⟨⟨_,student:name,student:id⟩⟩.
3. Rename the ⟨⟨student⟩⟩ node to ⟨⟨student:oid⟩⟩.
4. Use the inverse of identity node merge to extract a new ⟨⟨student⟩⟩ node with the same extent as ⟨⟨student:name⟩⟩, taking with it all the previous associations of ⟨⟨student:name⟩⟩. At this stage we will have the HDM shown in Figure 7.
5. Use inclusion merge to eliminate the ⟨⟨ug⟩⟩ subclass, and then use mandatory-unique direction to move ⟨⟨ug:ppt⟩⟩ to be associated with ⟨⟨student⟩⟩. Finally use

inverse of inclusion merge, and then inverse identity node merge to create the structure shown in Figure 3(b) for ⟨⟨ug⟩⟩ and ⟨⟨student⟩⟩.

At the end of this process, the ⟨⟨student:oid⟩⟩ node can be discarded, since it represents the object identifier of the UML model, that is not relevant in the ER model. A similar mapping process may be applied for the course and result classes. The result class will be mapped to a HDM form similar to the relational table result, which can be mapped to the ER result relation by the process already described at the start of this section. Note that this mapping shows that the semantic difference between the ER and UML models is that the ER model includes a statement of which attributes identify entities (which UML does not), and the UML model includes a hidden OID attribute (which the ER model does not have).

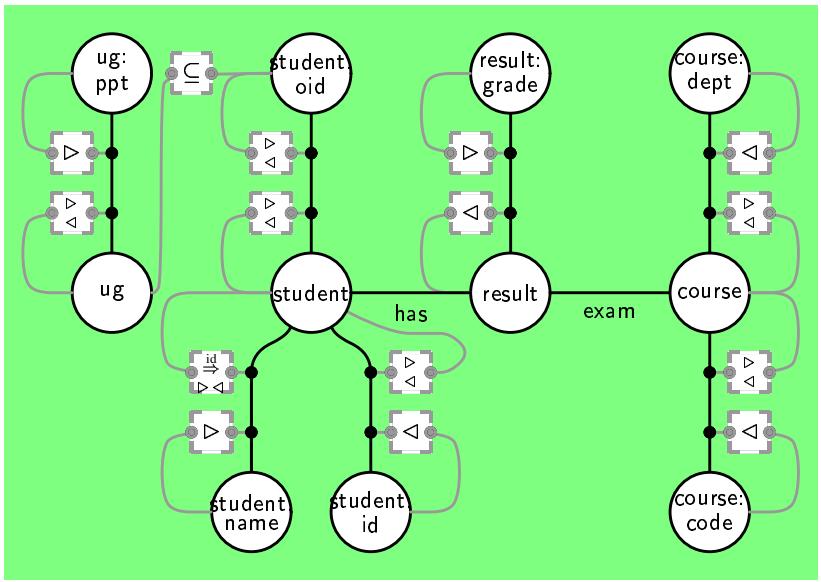


Fig. 7. UML to ER mapping after conversion of the student class

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 and ER models, and this will be extended to cover the ORM [6], object relational, YAT [3] (semi-structured), XML Schema data models.

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*, 2004.
3. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! *SIGMOD Record*, 27(2):177–188, 1998.
4. C.J. Date. Object indentifiers vs. relational keys. In *Relational Database: Selected Writings 1994–1997* [5].
5. C.J. Date, H. Darwen, and D. McGoveran. *Relational Database: Selected Writings 1994–1997*. Addison-Wesley, 1998.
6. T. Halpin. *Information Modeling and Relational Databases*. Academic Press, 2001.
7. E. Jasper, A. Poulovassilis, and L. Zamboulis. Processing IQL queries and migrating data in the AutoMed toolkit. Technical Report No. 20, AutoMed, 2003.
8. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.
9. 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.
10. P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.
11. 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.
12. 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.
13. 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.
14. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
15. K. Schewe. Design theory for advanced datamodels. In *Proc. 12th Australasian Conf. on Database Technologies*, pages 3–9, 2001.
16. R. Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, 1998.
17. C. Zaniolo and M. Melkanoff. A formal approach to the definition and the design of conceptual schemata for database systems. *ACM TODS*, 1982.