A Uniform Approach to Inter-Model Transformations

Peter McBrien and Alexandra Poulovassilis
Dept. of Computer Science, King's College London,
Strand, London WC2R 2LS
alex,pjm@dcs.kcl.ac.uk

Abstract

Whilst it is a common task in systems integration to have to transform between different semantic data models, such **inter-model transformations** are often specified in an ad hoc manner. Further, they are usually based on transforming all data into one common data model, which may not contain suitable data constructs to model directly all of the constructs of the data models being integrated. Our approach is to define each of the data models to be integrated in terms of a more 'elemental', low-level data model — a hypergraph-based one. We show how these definitions can be used to automatically derive schema transformation operators for the higher-level data models. These higher-level transformations can be used to perform inter-model transformations, and in turn allow data and queries to be automatically translated from one model to another. Finally, we show how to use the hypergraph data model in order to define inter-model links, and hence allow queries which span more than one model.

1 Introduction

Common to many areas of system integration, such as data warehousing, schema integration, and workflow systems, is the requirement to extract data associated with a particular **universe of discourse (UoD)** presented in one modelling language, and to use that data in another modelling language. Current approaches to mapping between such modelling languages usually choose one of them as the **common data model (CDM)** [15] and convert all the other modelling languages into that CDM.

Using a CDM such as the ER model or the relational model greatly complicates the mapping process, which requires that one high-level modelling language be specified in terms of another such language. This is due to the fact that there is rarely a simple correspondence between their modelling constructs. For example, if we use the relational model to represent ER models, a many-many relationship in the ER model must be represented as a relation in the relational model, whilst a one-many relationship can be represented as a foreign key attribute [7]. In the relational model, an attribute that forms part of a foreign key will be represented as a relationship in the ER model, whilst other relation attributes will be represented as ER attributes [1].

Our approach is to define a more 'elemental', low-level modelling language which is based on a hypergraph data structure together with a set of associated constraints, and to define a small set of primitive transformation operations on schemas expressed in this low-level notation. We then handle 'richer', higher-level modelling languages such as UML, ER, *etc*, by defining the constructs and transformations of these languages in terms of those of the hypergraph data model.

As will be seen in this paper, this enables a wide variety of modelling languages to be handled using some relatively simple definitions, and for transformation operations on schemas defined

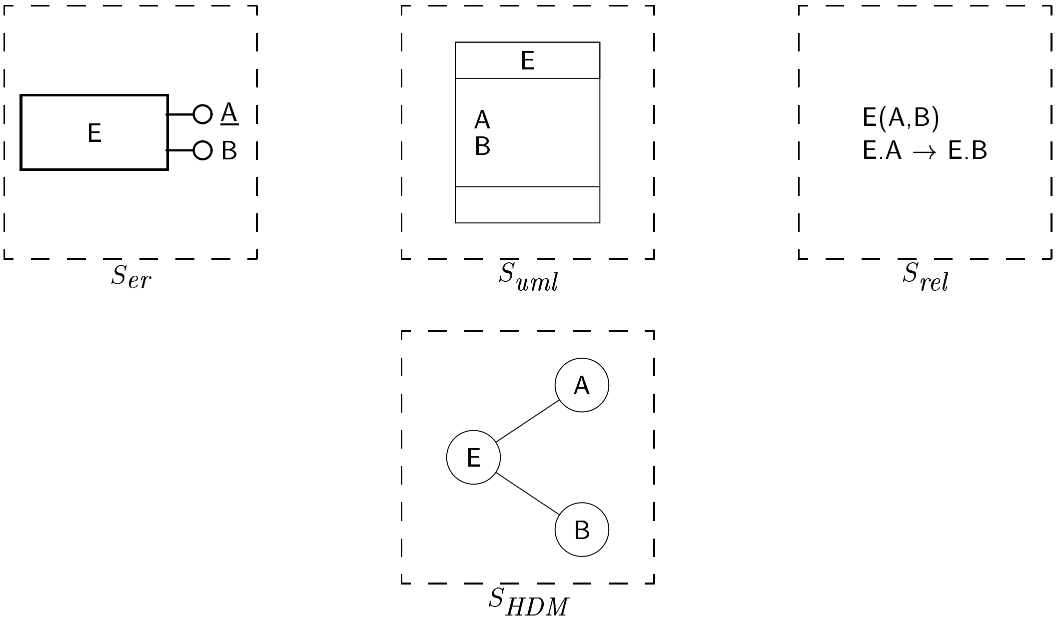

Figure 1: Multiple models based on the HDM

in these languages to be automatically derived from the language definition. This in turn allows **inter-model transformations** to be defined, *i.e.* transformation rules between constructs expressed in different modelling languages. These transformations can be used to migrate queries and data between schemas expressed in the different languages. Our approach has the advantage over **description logics** [5, 6] in that it clearly separates the modelling of a data structure from the modelling of constraints on instances of that data structure. However, in common with description logics, we can form a union of different modelling languages to model a certain UoD.

Our previous work [13, 10] has defined a framework for performing semantic **intra-model transformations**, where the original and transformed schema are represented in the same modelling language. In [13] we defined the notions of schemas and schema equivalence for the low-level **hypergraph data model (HDM)**. We gave a set of primitive transformations on HDM schemas that preserve schema equivalence, and we showed how more complex transformations may be formulated as sequences of these primitive transformations. We illustrated the expressiveness and practical usefulness of the framework by showing how the higher-level modelling constructs of a practical ER modelling language may be defined in terms of the HDM, and primitive transformations on ER schemas defined in terms of composite transformations on the equivalent HDM schemas. In [10] we showed how schema transformations that are automatically reversible can be used as the basis for the automatic migration of data and application logic between schemas expressed in the HDM or in an ER modelling language.

Here we extend that work in three ways. Firstly, we demonstrate how the constructs of relational data models, the static aspects of UML class diagrams, and WWW documents can also be handled within our framework. Secondly, we extend the framework to allow constructs from several modelling languages to exist together in one schema. Since all these constructs are defined in terms of the underlying hypergraph data model, this gives the foundation for the definition of inter-model transformations. Thirdly, we show how the work in [10] can now be used to automatically transform and migrate queries between schemas defined in different modelling languages.

The contribution of this paper is that we provide a formal yet practical approach to defining the semantics of modelling languages, leading to the automatic derivation of transformation rules. These rules may be applied by a user to map between semantically equivalent schemas expressed in the same or different modelling languages. Once the user has determined which transformation rules to use, these rules allow any data or queries on one schema to be automatically translated to the other schema. A further contribution is that our use of a unifying underlying data model allows for the definition of inter-model links, which support the development of stronger coupling between different modelling languages than is provided by current approaches.

The concept is illustrated in Figure 1, where we show three high-level schemas each of which has the same underlying lower-level HDM schema. For each of the three higher-level modelling languages (UML, ER and relational), we may reduce the concepts of entities, attributes, classes and relations to being *nodes* which are associated to each other by *edges* in the underlying low-level schema. The three schemas therefore have a common representation as a graph with three nodes and two edges, as well as some (unillustrated) constraints on the instances the graph may have.

The remainder of the paper is as follows. We begin with an overview of our low-level framework and the HDM in Section 2. In Section 3 we describe our general methodology for defining high-level modelling languages, and transformations for them, in terms of the low-level framework. We illustrate the approach by defining four specific modelling languages — an ER model, a relational model, UML static structure diagrams, and WWW documents. In Section 4 we show how to perform **inter-model transformations**, leading to the automated translation of queries between different modelling languages detailed in Section 5. In Section 6 we demonstrate how to use our approach to handle combinations of existing modelling languages, enhanced with **inter-model links**, and thus support queries spanning more than one modelling language. A summary of the paper and our conclusions are given in Section 7.

2 Overview of the Hypergraph Data Model

In this section we give a brief overview of those aspects of our previous work that are necessary for the purposes of this paper, and we refer the reader to [13, 10] for full details and formal definitions.

A **schema** in the Hypergraph Data Model (HDM) is a triple $\langle Nodes, Edges, Constraints \rangle$. A **query** q **over a schema** $S = \langle Nodes, Edges, Constraints \rangle$ is an expression whose variables are members of $Nodes \cup Edges$[1]. $Nodes$ and $Edges$ define a labelled, directed, nested hypergraph. It is nested in the sense that edges can link any number of both nodes *and* other edges. $Constraints$ is a set of boolean-valued queries over S. Nodes are uniquely identified by their names. Edges and constraints have an optional name associated with them.

An **instance** I of a schema $S = \langle Nodes, Edges, Constraints \rangle$ is a set of sets satisfying the following:

(i) each construct $c \in Nodes \cup Edges$ has an extent, denoted by $Ext_{S,I}(c)$, that can be derived from I; [2]

(ii) conversely, each set in I can be derived from the set of extents $\{Ext_{S,I}(c) \mid c \in Nodes \cup Edges\}$;

(iii) for each $e \in Edge$, $Ext_{S,I}(e)$ contains only values that appear within the extents of the constructs linked by e (*i.e.* domain integrity);

[1] Since what we provide is a framework, the query language is not fixed but will vary between different implementation architectures. In our examples in this paper, we assume that it is the relational calculus.

[2] Again, the language in which this derivation mapping is defined, and also that in point (ii), is not fixed by our framework.

(iv) the value of every constraint $c \in Constraints$ is true, the **value** of a query q being given by $q[c_1/Ext_{S,I}(c_1), \ldots, c_n/Ext_{S,I}(c_n)]$ where $c_1, \ldots, c_n$ are the constructs in $Nodes \cup Edges$.

We call the function $Ext_{S,I}$ an **extension mapping**, and $Range(Ext_{S,I})$ is the **extension** of the schema S w.r.t. the instance I. A **model** is a triple $\langle S, I, Ext_{S,I} \rangle$, where I is an instance of S and $Ext_{S,I}$ is an extension mapping from S to I.

Two schemas are **unconditionally equivalent (u-equivalent)** if they have the same set of instances. Given a condition f, a schema S **conditionally subsumes** a schema S' w.r.t. f if any instance of S' satisfying f is also an instance of S. Two schemas S and S' are **conditionally equivalent (c-equivalent)** w.r.t f if they each conditionally subsume each other w.r.t. f.

We first developed and explored these definitions of schemas, instances, and schema equivalence for an ER common data model, in earlier work [9, 11]. In particular, a detailed treatment and a comparison with other approaches to schema equivalence and schema transformation can be found in [11], which also discusses how our framework can be applied to schema integration.

We now list the primitive transformations of the HDM. Each transformation is a function that when applied to a model returns a new model. Each transformation has a proviso associated with it which states when the transformation is **successful**. Unsuccessful transformations return an "undefined" model, denoted by ϕ. Any transformation applied to ϕ returns ϕ.

1. $renameNode$ $\langle fromName, toName \rangle$ renames a node. Successful provided that $toName$ is not already the name of a node in the schema.
2. $renameEdge$ $\langle \langle fromName, c_1, \ldots, c_m \rangle, toName \rangle$ renames the edge named $fromName$ and linking schema constructs $c_1, \ldots, c_m$ to $toName$. Successful provided that $toName$ is not already the name of an edge in the schema.
3. $addConstraint$ c adds a new constraint c. Successful provided c evaluates to true.
4. $delConstraint$ c deletes a constraint. Successful provided c exists.
5. $addNode$ $\langle name, q \rangle$ adds a node named $name$ whose extent is given by the value of the query q. Successful provided a node of that name does not already exist.
6. $delNode$ $\langle name, q \rangle$ deletes a node. Here, q is a query that states how the extent of the deleted node could be recovered from the extents of the remaining schema constructs (thus, not violating property (ii) of an instance). Successful provided the node exists and participates in no edges.
7. $addEdge$ $\langle \langle name, c_1, \ldots, c_m \rangle, q \rangle$ adds a new edge between a sequence of existing schema constructs $c_1, \ldots, c_m$. The extent of the edge is given by the value of the query q. Successful provided the edge does not already exist, $c_1, \ldots, c_m$ exist, and q satisfies the appropriate domain constraints.
8. $delEdge$ $\langle \langle name, c_1, \ldots, c_m \rangle, q \rangle$ deletes an edge. The query q states how the extent of the deleted edge could be recovered from the extents of the remaining schema constructs. Successful provided the edge exists and participates in no edges.

For each of these transformations, we also define a version which takes as an extra argument a condition which must be satisfied in order for the transformation to be successful. A **composite transformation** is a sequence of $n \geq 1$ primitive transformations. If any one of these primitive transformations returns ϕ then so does the composite transformation overall.

A transformation t is **schema-dependent (s-d) w.r.t. a schema** S if t does not return ϕ for any model of S, otherwise t is **instance-dependent (i-d) w.r.t.** S. It is easy to see that if a schema S can be transformed to a schema S' by means of a s-d transformation, and vice versa, then S and S' are u-equivalent. Similarly, if a schema S can be transformed to a schema S' by means of an i-d transformation with proviso f, and vice versa, then S and S' are c-equivalent w.r.t f.

For example, if S consists of one node, employee, and S' consists of one node staff, then the transformation $rename$ $\langle \mathsf{employee}, \mathsf{staff} \rangle$ on S is s-d as is the transformation $rename$ $\langle \mathsf{staff}, \mathsf{employee} \rangle$ on S', and so S and S' are u-equivalent. On the other hand, if S consists of two nodes, employee and staff, and S' consists of one node staff, then the transformation $delNode$ $\langle \mathsf{employee}, \mathsf{staff} \rangle$ on S is i-d with proviso that $Ext_{S,I}(\mathsf{employee}) = Ext_{S,I}(\mathsf{staff})$ while the transformation $addNode$ $\langle \mathsf{employee}, \mathsf{staff} \rangle$ on S' is s-d. One of the transformations being i-d means that S and S' are c-equivalent w.r.t. the condition $Ext_{S,I}(\mathsf{employee}) = Ext_{S,I}(\mathsf{staff})$.

Every successful primitive transformation t is reversible by another successful primitive transformation t^{-1}, as follows:

Transformation (t)	Reverse Transformation (t^{-1})
$renameNode\ \langle from, to \rangle\ c$	$renameNode\ \langle to, from \rangle$
$renameEdge\ \langle \langle from, c_1, \ldots, c_m \rangle, to \rangle\ c$	$renameEdge\ \langle \langle to, c_1, \ldots, c_m \rangle, from \rangle$
$addConstraint\ q\ c$	$delConstraint\ q$
$delConstraint\ q\ c$	$addConstraint\ q$
$addNode\ \langle n, q \rangle\ c$	$delNode\ \langle n, q \rangle$
$delNode\ \langle n, q \rangle\ c$	$addNode\ \langle n, q \rangle$
$addEdge\ \langle \langle n, c_1, \ldots, c_m \rangle, q \rangle\ c$	$delEdge\ \langle \langle n, c_1, \ldots, c_m \rangle, q \rangle$
$delEdge\ \langle \langle n, c_1, \ldots, c_m \rangle, q \rangle\ c$	$addEdge\ \langle \langle n, c_1, \ldots, c_m \rangle, q \rangle$

This reversibility generalises to successful composite transformations: the reverse of the transformation $t_1; \ldots; t_n$ is $t_n^{-1}; \ldots; t_1^{-1}$.

3 Supporting Richer Semantic Modelling Languages

In this section we show how schemas expressed in higher-level semantic modelling languages, and the set of primitive transformations on such schemas, can be defined in terms of the hypergraph data model and its primitive transformations. We begin with a general discussion of how this is done for an arbitrary modelling language, M. We then illustrate the process for three specific modelling languages — an ER model, a relational model, and UML static structure diagrams. We conclude the section by also defining the conceptual elements of WWW documents, namely URL's, resources and links, and showing how these too can be represented in the HDM.

3.1 The General Approach

The constructs of any semantic modelling language M may be classified as either extensional constructs, or constraint constructs, or both. **Extentional constructs** represent sets of data values from some domain. Each such construct in M must be built using the extentional constructs of the HDM *i.e.* nodes and edges. There are three kinds of extentional constructs:

- **nodal** constructs may be present in a model independent of any other constructs. For example, ER model entities may be present without requiring the presence of any other particular constructs. A nodal construct maps into a node in the HDM. We use the following notation to define nodal constructs of M and their equivalent representation in the HDM, where the **scheme** of each construct uniquely identifies the construct in M:

Higher Level Construct	
Model	
Construct	
Classification	nodal
Scheme	

$\Rightarrow$

Equivalent HDM Representation	
Node	

- **linking** constructs may only be present in a model when certain other nodal constructs are already present. The extent of a linking construct is a subset of the Cartesian product of the extents of these nodal constructs. For example, relationships in ER models are linking constructs. Linking constructs map into edges in the HDM. We use the following notation to define linking constructs of M:

Higher Level Construct	
Model	
Construct	
Classification	linking
Scheme	

$\Rightarrow$

Equivalent HDM Representation	
Edge	
Linked-to	

- **nodal-linking** are like nodal constructs except that they can only be present in a model when certain other nodal constructs are already present, and are linked to these constructs. Attributes in ER models are an example of this. Nodal-linking constructs map into a combination of a node and an edge in the HDM. We use the following notation to define nodal-linking constructs of M:

Higher Level Construct	
Model	
Construct	
Classification	nodal-linking
Scheme	

$\Rightarrow$

Equivalent HDM Representation	
Node	
Edge	
Linked-to	

Constraint constructs represent restrictions on the extents of the extentional constructs of M. For example, ER model generalisation hierarchies restrict the extent of the subclass entities to be a subset of the extent of the superclass entity, and ER model relationships and attributes have cardinality constraints. Constraints are directly supported by the HDM, but if a constraint construct of M is also an extentional construct, then the appropriate extensional HDM constructs must also be included in its definition. We define constraint constructs of M using the following notation:

Higher Level Construct	
Model	
Construct	
Classification	constraint
Scheme	

$\Rightarrow$

Equivalent HDM Representation	
Linked-to	
Constraint	

The general method for constructing the set of primitive transformations on schemas of M is as follows:

(i) For every construct of M we need an *add* transformation to add to the underlying HDM schema the corresponding set of nodes, edges and constraints. This transformation thus consists of a sequence of 0 or one *addNode* transformations (the operand being taken from the

Node field of the construct definition), followed by a sequence of 0 or one *addEdge* transformations (taken from the Edge field), followed by a sequence of 0 or more *addConstraint* transformations (taken from the Constraint field). If the *addNode* or *addEdge* transformations are used, they should each be give a query expression which should appear in the scheme of the higher level construct.

(ii) For every construct of M we also need a *del* transformation which reverses the *add* transformation. This therefore consists of a sequence of *delConstraint* transformations, followed by a sequence of *delEdge* transformations, followed by a sequence of *delNode* transformations.

(iii) For those constructs of M which have textual names, we also define a *rename* transformation in terms of the corresponding set of *renameNode* and *renameEdge* transformations.

For example, attributes in ER Models (defined in 3.2.2 below) illustrate the method of deriving the *add* and *del* transformations on a construct from its HDM definition, while entities and relationships in ER Models (defined in 3.2.1 and 3.2.3) illustrate the method of deriving *rename* transformations. In particular, once a high-level construct has been defined in the HDM, the necessary *add*, *del* and *rename* transformations on it can be **automatically** derived from its HDM definition.

3.2 An ER Model

We first illustrate how our HDM framework can support a simple ER model with binary relationships and generalisation hierarchies (in [13] we show how the framework can support much richer ER models with n-ary relations, attributes on relations, and complex attributes, all of which we omit here for brevity). We also define a set of primitive transformations on such ER schemas and show how they can be defined in terms of the lower-level primitive transformations of the HDM.

We use some short-hand notation for expressing cardinality constraints on edges $\langle name, c_1, \ldots, c_m \rangle$ in that makeCard($\langle name, c_1, \ldots, c_m \rangle, s_1, \ldots, s_m$) denotes the following constraint:

$$\bigwedge_{i=1}^{m} (\forall x \in c_i \,.\, |\{\langle v_1, \ldots, v_m \rangle \mid \langle v_1, \ldots, v_m \rangle \in \langle name, c_1, \ldots, c_m \rangle \wedge v_i = x\}| \in s_i)$$

Here, each s_i is a set of integers representing the possible values for the cardinality of the construct c_i in the edge $\langle name, c_1, \ldots, c_m \rangle$ e.g. $\{0, 6..10, 20..N\}$, N denoting infinity. Note that [8] identified this notation as the most expressive method for specifying cardinality constraints, in a survey of a large number of static modelling languages, and therefore it should be sufficient for modelling all conceptual modelling languages we are likely to encounter.

3.2.1 Entity

Entity classes in ER schemas map to nodes in the underlying HDM schema. Because we will later be mixing schema constructs from schemas that may be expressed in different modelling notations, we disambiguate these constructs at the HDM level by adding a prefix to their name. This prefix is *er*, *rel*, *uml* or *www* for each of the four modelling notations that we will be considering.

Higher Level Construct	
Model	er
Construct	entity
Classification	nodal
Scheme	$\langle e \rangle$

$\Rightarrow$

Equivalent HDM Representation	
Node	$\langle er{:}e \rangle$

Transformation on er	Equivalent Transformation on HDM
$rename^{er}_E\langle e, e'\rangle$	$renameNode\ \langle er{:}e, er{:}e'\rangle$
$add^{er}_E\langle e, q\rangle$	$addNode\ \langle er{:}e, q\rangle$
$del^{er}_E\langle e, q\rangle$	$delNode\ \langle er{:}e, q\rangle$

3.2.2 Attribute

Attributes in ER schemas also map to nodes in the HDM, since they have an extent. However, attributes must always be linked to entities, and hence are classified as nodal-linking. The cardinality constraints on attributes lead to them being classified also as constraint constructs. Note that in the HDM schema we prefix the name of the attribute by its entity's name, so that we can regard as distinct two attributes with the same name if they are are attached to different entities. The association between an entity and an attribute is un-named, hence the occurrence of _ in the equivalent HDM edge construct.

Higher Level Construct	
Model	er
Construct	attribute
Classification	nodal-linking, constraint
Scheme	$\langle e, a, s_1, s_2\rangle$

⇒

Equivalent HDM Representation	
Node	$\langle er{:}e{:}a\rangle$
Edge	$\langle \_, er{:}e, er{:}e{:}a\rangle$
Linked-to	$\langle er{:}e\rangle$
Constraint	makeCard$(\langle _, er{:}e, er{:}e{:}a\rangle, s_1, s_2)$

Transformation on er	Equivalent Transformation on HDM
$rename^{er}_A\langle a, a'\rangle$	$renameNode\ \langle er{:}e{:}a, er{:}e{:}a'\rangle$
$add^{er}_A\langle e, a, s_1, s_2, q_{att}, q_{assoc}\rangle$	$addNode\ \langle er{:}e{:}a, q_{att}\rangle$; $addEdge\ \langle\langle \_, er{:}e, er{:}e{:}a\rangle, q_{assoc}\rangle$; $addConstraint$ makeCard$(\langle _, er{:}e, er{:}e{:}a\rangle, s_1, s_2)$
$del^{er}_A\langle e, a, s_1, s_2, q_{att}, q_{assoc}\rangle$	$delConstraint$ makeCard$(\langle _, er{:}e, er{:}e{:}a\rangle, s_1, s_2)$; $delEdge\ \langle\langle \_, er{:}e, er{:}e{:}a\rangle, q_{assoc}\rangle$; $delNode\ \langle er{:}e{:}a, q_{att}\rangle$

By making the query q_{att} specifying the extent of an attribute to be just $\{y \mid \exists x.\langle x, y\rangle \in q_{assoc}\}$, we force attribute instances to only exist when they are associated to an entity instance. Thus it is not necessary to use q_{att} as an argument to the add^{er}_A and del^{er}_A transformations since it can be derived from q_{assoc}. However, we include it here for ease of readability of the underlying HDM transformations.

3.2.3 Relationship

Relationships in ER schemas map to edges in the HDM and are classified linking constructs. As with attributes, the cardinality constraints on relationships lead to them being classified also as constraint constructs.

Higher Level Construct	
Model	er
Construct	relationship
Classification	linking, constraint
Scheme	$\langle r, e_1, e_2, s_1, s_2\rangle$

⇒

Equivalent HDM Representation	
Edge	$\langle er{:}r, er{:}e_1, er{:}e_2\rangle$
Linked-to	$\langle er{:}e_1\rangle, \langle er{:}e_2\rangle$
Constraint	makeCard$(\langle er{:}r, er{:}e_1, er{:}e_2\rangle, s_1, s_2)$

Transformation on er	Equivalent Transformation on HDM
$rename^{er}_R\langle\langle r, e_1, e_2\rangle, r'\rangle$	$renameEdge\ \langle\langle er{:}r, er{:}e_1, er{:}e_2\rangle, er{:}r'\rangle$
$add^{er}_R\langle r, e_1, e_2, s_1, s_2, q\rangle$	$addEdge\ \langle\langle er{:}r, er{:}e_1, er{:}e_2\rangle, q\rangle$; $addConstraint$ makeCard$(\langle er{:}r, er{:}e_1, er{:}e_2\rangle, s_1, s_2)$
$del^{er}_R\langle r, e_1, e_2, s_1, s_2, q\rangle$	$delConstraint$ makeCard$(\langle er{:}r, er{:}e_1, er{:}e_2\rangle, s_1, s_2)$; $delEdge\ \langle\langle er{:}r, er{:}e_1, er{:}e_2\rangle, q\rangle$

3.2.4 Generalisation

ER model generalisations are constraints on the instances of entity classes, which we give a textual name to. We use the notation *label*[*cons*] to denote a labelled constraint in the HDM, and provide the additional primitive transformation $renameConstraint\langle label, label'\rangle$. Several constraints may have the same label, indicating that they are associated with the same higher-level schema construct. Generalisations in ER models are uniquely identified by the combination of the superclass entity name, e, and the generalisation name, g, so we use the pair $e{:}g$ as the label for the constraints associated with a generalisation. Generalisations may be partial or total.

Higher Level Construct	
Model	er
Construct	generalisation
Classification	constraint
Scheme	$\langle pt, e, g, e_1, \ldots, e_n\rangle$

⇒

Equivalent HDM Representation	
Linked-to	$\langle er{:}e\rangle, \langle er{:}e_1\rangle, \ldots, \langle er{:}e_n\rangle$
Constraint	$e{:}g[\forall 1 \leq i < j \leq n \,.\, e_i \cap e_j = \emptyset]$; $e{:}g[\forall 1 \leq i \leq n \,.\, e_i \subseteq e]$; if $pt = \mathsf{total}$ then $e{:}g[e = \bigcup_{i=1}^{n} e_i]$

Transformation on er	Equivalent Transformation on HDM
$rename^{er}_G\langle e, g, g'\rangle$	$renameConstraint\ \langle er{:}e{:}g, er{:}e{:}g'\rangle$
$add^{er}_G\langle pt, e, g, e_1, \ldots, e_n\rangle$	if $pt = \mathsf{total}$ then $addConstraint\ e{:}g[e = \bigcup_{i=1}^{n} e_i]$; $addConstraint\ e{:}g[\forall 1 \leq i \leq n \,.\, e_i \subseteq e]$; $addConstraint\ e{:}g[\forall 1 \leq i < j \leq n \,.\, e_i \cap e_j = \emptyset]$
$del^{er}_G\langle pt, e, g, e_1, \ldots, e_n\rangle$	if $pt = \mathsf{total}$ then $delConstraint\ e{:}g[e = \bigcup_{i=1}^{n} e_i]$; $delConstraint\ e{:}g[\forall 1 \leq i < j \leq n \,.\, e_i \cap e_j = \emptyset]$; $delConstraint\ e{:}g[\forall 1 \leq i \leq n \,.\, e_i \subseteq e]$

To simplify the specification of different variants of the same transformation, we have here introduced the concept of a **conditional template transformation**. This takes the form 'if q_{cond} then t' where q_{cond} is a query over the schema component of the model that the transformation is being applied to. q_{cond} may contain free variables that are instantiated by the transformation's arguments. If q_{cond} evaluates to true, then those instantiations substitute for the same free variables in t, which forms the result of the template. If q_{cond} evaluates false, then the result of the template is the identity transformation. The templates may be extended with an else clause, of the form 'if q_{cond} then t else t'', where if q_{cond} is false then the result is t'.

3.3 The Relational Model

We now show how to represent the basic relational data model in the HDM, where we take the relational model to consist of relations, attributes (which may be null), a primary key for each relation, and foreign keys. Due to space constraints, our descriptions for this model and for the following UML and WWW Document Models omit the definitions of the primitive transformation operations on these models. These primitive transformations can be automatically derived from the definition of each modelling construct by the general method already described in Section 3.1 above (steps (i)-(iii)) and illustrated for the ER model in Section 3.2 above.

3.3.1 Relation

Relations may exist independently of each other and are nodal constructs. Normally relational languages do not allow the user to query the extent of a relation (but rather the attributes of the relation) so we choose here to define the extent of the relation by its primary key, to simplify the definition of the remaining constructs.

Higher Level Construct	
Model	rel
Construct	relation
Classification	nodal
Scheme	$\langle r \rangle$

$\Rightarrow$

Equivalent HDM Representation	
Node	$\langle rel{:}r \rangle$

3.3.2 Attributes

The attributes of relations in the relational model are similar to the attributes of entity classes in the ER model. However, the cardinality constraint is now a simple choice between the attribute being optional ($\mathsf{null} \to \{0,1\}$) or mandatory ($\mathsf{notnull} \to \{1\}$).

Higher Level Construct	
Model	rel
Construct	attribute
Classification	nodal-linking, constraint
Scheme	$\langle r, a, n \rangle$

$\Rightarrow$

Equivalent HDM Representation	
Node	$\langle rel{:}r{:}a \rangle$
Edge	$\langle \_, rel{:}r, rel{:}r{:}a \rangle$
Linked-to	$\langle rel{:}r \rangle$
Constraint	if $(n = \mathsf{null})$ then makeCard($\langle \_, rel{:}r, rel{:}r{:}a \rangle, \{0,1\}, \{1..N\}$) else makeCard($\langle \_, rel{:}r, rel{:}r{:}a \rangle, \{1\}, \{1..N\}$)

3.3.3 Primary Key

Since we represent the extent of a relation by its set of primary key attributes, the primary key amounts to a constraint checking that the extent of r is the same as the extent of the tuple made from the key attributes $a_1, \ldots, a_n$.

Higher Level Construct	
Model	rel
Construct	primary key
Classification	constraint
Scheme	$\langle r, a_1, \ldots, a_n \rangle$

$\Rightarrow$

Equivalent HDM Representation	
Linked-to	$\langle rel{:}r{:}a_1 \rangle, \ldots, \langle rel{:}r{:}a_n \rangle$
Constraint	$x \in \langle rel{:}r \rangle \leftrightarrow x = \langle x_1, \ldots, x_n \rangle$ $\wedge \langle x, x_1 \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_1 \rangle$ $\vdots$ $\wedge \langle x, x_n \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_n \rangle$

3.3.4 Foreign Key

Here, attributes $a_1, \ldots, a_n$ appearing in r are the primary key of some relation r_f:

Higher Level Construct	
Model	rel
Construct	foreign key
Classification	constraint
Scheme	$\langle r, r_f, a_1, \ldots, a_n \rangle$

$\Rightarrow$

Equivalent HDM Representation	
Linked-to	$\langle rel{:}r{:}a_1 \rangle, \ldots, \langle rel{:}r{:}a_n \rangle$
Constraint	$\langle x, x_1 \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_1 \rangle \wedge$ $\vdots$ $\langle x, x_n \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_n \rangle \wedge$ $\rightarrow \langle x_1, \ldots, x_n \rangle \in r_f$

3.4 UML Static Structure Diagrams

We define now the elements of UML class diagrams that model static aspects of the UoD. The elements of class diagrams that are identified as dynamic in the UML Notation Guide [4] are not covered here. Thus we do not model operations, nor the distinction between associations and composition.

3.4.1 Class

Higher Level Construct	
Model	uml
Construct	class
Classification	nodal
Scheme	$\langle c \rangle$

$\Rightarrow$

Equivalent HDM Representation	
Node	$\langle uml{:}c \rangle$

3.4.2 MetaClass

UML metaclasses are defined in the same manner as classes, with the constraint that the instances of a metaclass must themselves be classes.

Higher Level Construct	
Model	uml
Construct	entity
Classification	nodal, constraint
Scheme	$\langle m \rangle$

$\Rightarrow$

Equivalent HDM Representation	
Node	$\langle uml{:}m \rangle$
Constraint	$c \in \langle uml{:}m \rangle \rightarrow \langle c \rangle$

3.4.3 Attribute

UML attributes have a **multiplicity** associated with them. This is a single range of integers which shows how many instances of the attribute each instance of the entity may be associated with. Below we represent this by a single cardinality constraint, s. The cardinality constraint on the attribute is by definition $[1..N]$.

Higher Level Construct	
Model	uml
Construct	attribute
Classification	nodal-linking, constraint
Scheme	$\langle c, a, s\rangle$

$\Rightarrow$

Equivalent HDM Representation	
Node	$\langle uml{:}c{:}a\rangle$
Edge	$\langle \_, uml{:}c, uml{:}c{:}a\rangle$
Linked-to	$\langle uml{:}c\rangle$
Constraint	makeCard$(\langle _, uml{:}c, uml{:}c{:}a\rangle, s, \{1..N\})$

Note that we do not restrict the domain of a in any way, so we are able to use the above definition to support attributes which have either simple types or entity classes as their domain. A more elaborate implementation could add a field to the scheme of each attribute to indicate the domain from which values of the attribute are drawn. To keep the presentation concise we avoid entering into that detail in this paper.

3.4.4 Objects

An **object** in the UML notation constrains the instances of some class, in the sense that the class must have at least one instance where the attributes $a_1, \ldots, a_n$ take specified values $v_1, \ldots, v_n$. Hence a UML object is a constraint construct, labelled with the name of the object, o, and the class, c, it is an instance of.

Higher Level Construct	
Model	uml
Construct	object
Classification	constraint
Scheme	$\langle c, o, a_1, \ldots, a_n, v_1, \ldots, v_n\rangle$

$\Rightarrow$

Equivalent HDM Representation	
Linked-to	$\langle uml{:}c\rangle, \langle uml{:}c, uml{:}a_1\rangle, \ldots, \langle uml{:}c, uml{:}a_n\rangle$
Constraint	$c{:}o[{}^{\exists i \in uml{:}c\,.\,\langle i, v_1\rangle \in \langle uml{:}c, uml{:}a_1\rangle}_{\wedge\ \ldots \wedge \langle i, v_n\rangle \in \langle uml{:}c, uml{:}a_n\rangle}]$

3.4.5 Association and Compositions

UML supports both binary and n-ary associations. Binary association is just a special case of n-ary association [4], with is own notation. Thus we only consider here the general case of n-ary associations, in which an association r links classes $c_1, \ldots, c_n$, with role names $l_1, \ldots, l_n$ and cardinalities of each role $s_1, \ldots, s_n$. Note that we identify the association in the HDM by concatenating the association name with the role names.

Higher Level Construct	
Model	uml
Construct	association
Classification	linking, constraint
Scheme	$\langle r, c_1, \ldots, c_n, l_1, \ldots, l_n, s_1, \ldots, s_n\rangle$

$\Rightarrow$

Equivalent HDM Representation	
Edge	$\langle uml{:}r{:}l_1{:}\ldots{:}l_n, uml{:}c_1, \ldots, uml{:}c_n\rangle$
Linked-to	$\langle uml{:}c_1\rangle, \ldots, \langle uml{:}c_n\rangle$
Constraint	makecard$(\langle uml{:}r{:}l_1{:}\ldots{:}l_n, uml{:}c_1, \ldots, uml{:}c_n\rangle, s_1, \ldots, s_n)$

UML's **composition** construct is a restriction of an association to have $\{1\}$ cardinality on the number of instances of the parent class that each instance of the child class can be associated with. together with certain restrictions on the dynamic behaviour of these classes. We can therefore model the static aspects of compositions by associations, specifying the cardinality constraint $\{1\}$ in the Scheme entry above.

3.4.6 Generalisation

UML generalisations may be either incomplete or complete, and overlapping or disjoint [4]. Hence we have two template transformations to handle these two distinctions.

Higher Level Construct	
Model	uml
Construct	generalisation
Classification	constraint
Scheme	$\langle cs, c, g, c_1, \ldots, c_n \rangle$

⇒

Equivalent HDM Representation	
Linked-to	$\langle uml{:}c \rangle, \langle uml{:}c_1 \rangle, \ldots, \langle uml{:}c_n \rangle$
Constraint	if disjoint $\in cs$ then $c{:}g[\forall 1 \leq i < j \leq n \,.\, c_i \cap c_j = \emptyset]$; if complete $\in cs$ then $c{:}g[c = \bigcup_{i=1}^{n} c_i]$; $c{:}g[\forall 1 \leq i \leq n \,.\, c_i \subseteq c]$

3.5 WWW Documents

Before describing how WWW Documents are represented in the HDM, we first identify how they can be structured as conceptual elements. URLs [2] for Internet resources fetched using the IP protocol from specific hosts take following general form:

⟨scheme⟩://⟨user⟩:⟨password⟩@⟨host⟩:⟨port⟩/⟨url-path⟩

We can therefore characterise a URL as an HDM node, formed of sextuples consisting of these six elements of the URL (with _ used for missing values of optional elements). A WWW document resource can be modelled as another node. Each resource must be identified by a URL, but a URL may exist without its corresponding resource existing. Each resource may link to any number of URLs. Thus we have a single HDM schema for the WWW which is constructed as follows:

addNode ⟨*www:*URL,{}⟩;
addNode ⟨*www:*resource,{}⟩;
addEdge ⟨⟨*www:*identifies,*www:*URL,*www:*resource⟩,{0..1},{1},{}⟩;
addEdge ⟨⟨*www:*link,*www:*resource,*www:*URL⟩,{0..N},{0..N},{}⟩;

Notice that we have assigned an empty extent to each of the four extensional constructs of the WWW schema. This is because we model each URL, resource or link in the WWW as a **constraint construct**, requiring the existence of this instance in the extension of the WWW schema. These constructs therefore generate no transformations for adding or deleting HDM nodes or edges, only transformations for adding or deleting the constraint associated with the instance.

3.5.1 URLs

Higher Level Construct	
Model	www
Construct	url
Classification	constraint
Scheme	$\langle s, us, pw, h, pt, up \rangle$

⇒

Equivalent HDM Representation	
Linked-to	$\langle www{:}\text{url} \rangle$
Constraint	$\langle s, us, pw, h, pt, up \rangle \in \langle www{:}\text{url} \rangle$

3.5.2 Resources

Higher Level Construct	
Model	www
Construct	resource
Classification	constraint
Scheme	$\langle\langle s, us, pw, h, pt, up\rangle, r\rangle$

⇒

Equivalent HDM Representation	
Linked-to	$\langle www{:}\mathsf{url}\rangle$
Constraint	$\langle\langle s, us, pw, h, pt, up\rangle, r\rangle \in$ $\langle www{:}\mathsf{identifies}, www{:}\mathsf{url}, www{:}\mathsf{resource}\rangle$

3.5.3 Links

Higher Level Construct	
Model	www
Construct	link
Classification	constraint
Scheme	$\langle r, \langle s, us, pw, h, pt, up\rangle\rangle$

⇒

Equivalent HDM Representation	
Linked-to	$\langle www{:}\mathsf{url}\rangle$
Constraint	$\langle r, \langle s, us, pw, h, pt, up\rangle\rangle \in$ $\langle www{:}\mathsf{link}, www{:}\mathsf{resource}, www{:}\mathsf{url}\rangle$

4 Inter-Model Transformations

We observe from the previous section that our representation of higher-level modelling languages in the HDM is such that it is possible to unambiguously represent the constructs of multiple higher-level schemas **simultaneously** in one HDM schema. This brings several important benefits:

(a) An HDM schema can be used as a unifying repository for several higher-level schemas.

(b) Add and delete transformations can be carried out for constructs of a modelling language M_1 where the extent of the construct is defined in terms of the extents of constructs of some other modelling languages, $M_2, M_3, \ldots$. This allows us to perform **inter-model transformations** where the constructs of one modelling language are replaced with those of another.

(c) Such inter-model transformations form the basis for automatic **inter-model translation** of both data and queries between different models. This facility allows data and queries to be translated between different schemas in interoperating database architectures such as database federations [15] and mediators [16].

(d) New **inter-model edges** which do not belong to any single higher-level modelling language can be defined. This allows us to build associations between constructs in different modelling languages and to navigate between them. This facility is of particular use when no single higher-level modelling language can adequately capture the UoD, as is invariably the case with any large complex application domain.

We further elaborate on item (b) here and on item (c) in Section 5. Items (a) and (d) are discussed further in Section 6.

We use the syntax $M(q)$ to indicate that a query q should be evaluated with respect to the schema constructs of the higher-level model M, where M can be rel, er, uml, www and so forth. If q appears in the argument list of a transformation on a construct of M, then it may be written simply as q, and $M(q)$ is inferred. For example, $add_C^{uml}\langle\mathsf{person}, \mathsf{student}\rangle$ is equivalent to $add_C^{uml}\langle\mathsf{person}, uml(\mathsf{student})\rangle$, meaning add a UML class person, and define the extent of that class to be the same as that of UML class student, while $add_C^{uml}\langle\mathsf{person}, er(\mathsf{student})\rangle$ would populate the instances of class person with the instances of ER entity student (which is unlikely to be semantically correct, but serves to illustrate the point).

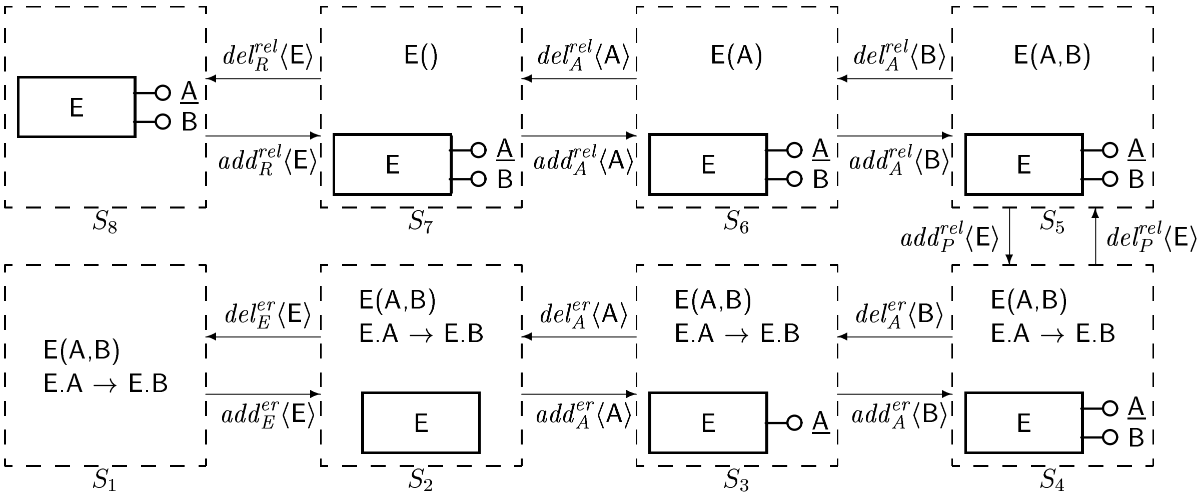

Figure 2: Inter-model transformation example

Example 1 A Relational to ER inter-model transformation

Figure 2 illustrates the following transformation of a relational schema S_1 to an ER schema S_8. $rel(q)$ denotes a query q that should be evaluated with respect to the relational schema constructs, and $er(q)$ a query that should be evaluated with respect to the ER schema constructs. getCard(c) denotes the cardinality constraint associated with a construct c.

1. $add_E^{er}\langle \mathsf{E}, rel(\{y \mid \exists x.\langle x,y\rangle \in \langle \_, \mathsf{E}, \mathsf{A}\rangle\})\rangle$
2. $add_A^{er}\langle \mathsf{E}, \mathsf{A}, \mathrm{getCard}(\langle \_, rel{:}\mathsf{E}, rel{:}\mathsf{E}{:}\mathsf{A}\rangle), rel(\{y \mid \exists x.\langle x,y\rangle \in \langle \_, \mathsf{E}, \mathsf{A}\rangle\}), rel(\langle \_, \mathsf{E}, \mathsf{A}\rangle)\rangle$
3. $add_A^{er}\langle \mathsf{E}, \mathsf{B}, \mathrm{getCard}(\langle \_, rel{:}\mathsf{E}, rel{:}\mathsf{E}{:}\mathsf{B}\rangle), rel(\{y \mid \exists x.\langle x,y\rangle \in \langle \_, \mathsf{E}, \mathsf{B}\rangle\}), rel(\langle \_, \mathsf{E}, \mathsf{B}\rangle)\rangle$
4. $del_P^{rel}\langle \mathsf{E}, \mathsf{A}\rangle$
5. $del_A^{rel}\langle \mathsf{E}, \mathsf{B}, \mathrm{getCard}(\langle \_, er{:}\mathsf{E}, er{:}\mathsf{E}{:}\mathsf{B}\rangle), er(\{y \mid \exists x.\langle x,y\rangle \in \langle \_, \mathsf{E}, \mathsf{B}\rangle\}), er(\langle \_, \mathsf{E}, \mathsf{B}\rangle)\rangle$
6. $del_A^{rel}\langle \mathsf{E}, \mathsf{A}, \mathrm{getCard}(\langle \_, er{:}\mathsf{E}, er{:}\mathsf{E}{:}\mathsf{A}\rangle), er(\{y \mid \exists x.\langle x,y\rangle \in \langle \_, \mathsf{E}, \mathsf{A}\rangle\}), er(\langle \_, \mathsf{E}, \mathsf{A}\rangle)\rangle$
7. $del_R^{rel}\langle \mathsf{E}, er(\{y \mid \exists x.\langle x,y\rangle \in \langle \_, \mathsf{E}, \mathsf{A}\rangle\})\rangle$

Notice that the reverse transformation from S_8 back to S_1 is automatically derivable, as described at the end of Section 2, and we have illustrated it too in Figure 2.

□

Whilst it is possible to write inter-model transformations such as the one illustrated in Example 1 for each specific transformation as it arises, this can be tedious and repetitive, and in practice we will want to automate the process. We can use template transformations to specify in a generic way how constructs in a modelling language M_1 should be transformed to constructs in a modelling language M_2, thus enabling transformations on specific constructs to be automatically generated. The following guidelines can be used in preparing the template transformations:

1. Ensure that every possible instance of any construct in M_1 appears in the query part of a transformation that adds a construct to M_2.

   Occasionally it might be possible to consider these instances individually, such as in the first transformation step of Example 2. However usually it is combinations of instances of constructs in M_1 that map to instances of constructs in M_2, and the rest of the transformation steps in that example illustrate.

2. Ensure that every construct c of M_1 appears in a transformation that deletes c from M_1, recovering the extent of c from the extents of constructs in M_2 that were created in one of the addition transformations in the previous step above.

We illustrate the first step in this process by showing how the constructs of a relational model can be mapped to constructs of an ER model in Example 2. We recall that $rel(q)$ denotes a query q that should be evaluated with respect to the relational schema constructs, and $er(q)$ a query that should be evaluated with respect to the ER schema constructs.

Example 2 Mapping Relational Models to ER Models

1. Each relation r can be represented as a entity class with the same name in the ER model, using its primary key to identify instances of the entity class.

   if $\langle r, \_, \ldots, \_\rangle \in$ primarykey
   then $add^{er}_{E}\langle r, rel(\langle r\rangle)\rangle$

2. An attribute set of r which is its primary key and is also a foreign key which is the primary key of r_f, can be represented in the ER Model as a partial generalisation hierarchy with r_f as the superclass of r:

   if $\langle r, a_1, \ldots, a_n\rangle \in$ primarykey $\wedge \langle r, r_f, a_1, \ldots, a_n\rangle \in$ foreignkey
   then $add^{er}_{G}\langle \mathsf{partial}, r_f, r\rangle$

3. An attribute set of r which is not its primary key but which is a foreign key that is the primary key of r_f, can be presented in the ER Model as a relationship between r and r_f.

   if $\langle r, b_1, \ldots, b_n\rangle \in$ primarykey $\wedge \langle r, r_f, a_1, \ldots, a_n\rangle \in$ foreignkey $\wedge \{a_1, \ldots, a_n\} \neq \{b_1, \ldots, b_n\}$
   then $add^{er}_{R}\langle a_1{:}\ldots{:}a_n, r, r_f, \{0, 1\}, \{0..N\}, \{\langle\langle y_1, \ldots, y_n\rangle, \langle x_1, \ldots, x_n\rangle\rangle \mid \exists x\, .$
   $\langle x, y_1\rangle \in rel(\langle r, b_1\rangle) \wedge \ldots \wedge \langle x, y_n\rangle \in rel(\langle r, b_n\rangle) \;\wedge$
   $\langle x, x_1\rangle \in rel(\langle r, a_1\rangle) \wedge \ldots \wedge \langle x, x_n\rangle \in rel(\langle r, a_n\rangle)\}\rangle$

4. Any attribute of r that is not part of a foreign key can be represented as a ER Model attribute.

   if $\langle r, a\rangle \in$ attribute $\wedge \neg\exists a_1, \ldots, a_n\, .\, (\langle r, \_, a_1, \ldots, a_n\rangle \in$ foreignkey $\wedge\, a \in \{a_1, \ldots, a_n\})$
   then $add^{er}_{A}\langle r, a, \{0, 1\}, \{1..N\}, \{x \mid \exists y\, .\, \langle y, x\rangle \in rel(\langle r, a\rangle)\}, rel(\langle r, a\rangle)\rangle$

□

5 Inter-Model Queries

The reversibility of the HDM primitive transformations, which we discussed in Section 2, means that data and queries can be automatically translated between the original and the transformed schema. Since higher-level transformations are defined in terms of HDM transformations, these too will be automatically reversible, and so data and queries expressed in higher-level formalisms are also automatically translatable. This automatic translation of data and queries is discussed in detail in [10] and we only briefly summarise the translation method here.

Suppose we have a transformation that transforms a schema S_1 to a schema S_2. To translate the extension of S_1 to S_2 we need only to consider those transformation steps that add a new node or edge in the underlying HDM:

- $addNode$ $\langle n, q\rangle$: the new node n is populated by evaluating q on the extension of S_1.
- *addEdge* $\langle e, q\rangle$: the new edge e is similarly populated by evaluating q on the extension of S_1.

To translate a query q_1 posed on S_1 to an equivalent query q_2 posed on S_2 we need only to consider those transformation steps that rename or delete a node or edge. In these cases, we substitute old names by new names, and occurrences of deleted nodes or edges by their restoring query:

- $renameNode$ $\langle from, to\rangle$: $q_2 = [from/to]q_1$.
- $renameEdge$ $\langle\langle from, c_1, \ldots, c_m\rangle, to\rangle$: $q_2 = [\langle from, c_1, \ldots, c_m\rangle/\langle to, c_1, \ldots, c_m\rangle]q_1$.
- $delNode$ $\langle n, q\rangle$: $q_2 = [n/q]q_1$.
- *delEdge* $\langle e, q\rangle$: $q_2 = [e/q]q_1$.

For composite transformations, these substitutions are successively composed in order to obtain the final migrated query q_2.

We can now use this method to migrate data and queries between schemas expressed in different modelling languages. For example, considering the schemas S_1 and S_8 in Example 1 and Figure 2, we might have the following query on S_8 which finds the distinct instances of the E entity which share the same value for the B attribute (we use the $er(q)$ notation here to make it explicit that the named constructs come from the er model):

$$\langle x_1, y\rangle \in er(\langle \_, \mathsf{E}, \mathsf{B}\rangle) \wedge \langle x_2, y\rangle \in er(\langle \_, \mathsf{E}, \mathsf{B}\rangle) \wedge x_1 \neq x_2$$

The fifth transformation step from S_8 to S_1 ($del^{er}_{A}\langle \mathsf{B}\rangle$ operating on S_4 to give S_3) is automatically derived from the third transformation step in Example 1 ($add^{er}_{A}\langle \mathsf{B}\rangle$ operating on S_3 to give S_4). It gives rise to the following substitution:

$$[er(\langle \_, \mathsf{E}, \mathsf{B}\rangle)/rel(\langle \_, \mathsf{E}, \mathsf{B}\rangle)]$$

Applying this substitution to the above query on S_8 we obtain the following equivalent relational query on S_1:

$$\langle x_1, y\rangle \in rel(\langle \_, \mathsf{E}, \mathsf{B}\rangle) \wedge \langle x_2, y\rangle \in rel(\langle \_, \mathsf{E}, \mathsf{B}\rangle) \wedge x_1 \neq x_2$$

Here the translation is a very simple one, but the same procedure can clearly be used for more complex queries and several examples are given in [10].

6 Mixed Models

Our framework opens up the possibility of creating special-purpose CDMs which mix constructs from different modelling languages. This will be particularly useful in integration situations where there is not a single already existing CDM that can fully represent the constructs of the various

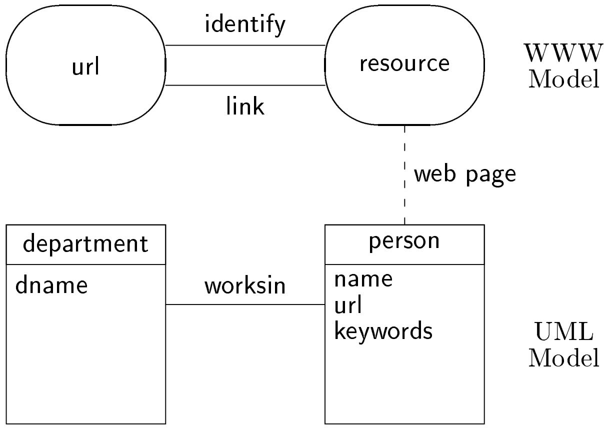

Figure 3: Linking a semantic data model with the WWW

data sources. To allow the user to navigate between the constructs of different modelling languages, we can specify inter-model edges that connect associated constructs.

For example, we may want to associate entity classes in an ER model with UML classes in a UML model, using a certain attribute in the UML model to correspond with the key attribute of the ER class. Based on this principle, we could define a new construct *common class* as follows:

Higher Level Construct	
Model	inter
Construct	common class
Classification	linking, constraint
Scheme	$\langle e, c, a\rangle$

$\Rightarrow$

Equivalent HDM Representation	
Edge	$\langle \_, \mathit{er{:}e}, \mathit{uml{:}c}\rangle$
Linked-to	$\langle \mathit{er{:}e}\rangle, \langle \mathit{uml{:}c}\rangle$
Constraint	$\langle \_, \mathit{er{:}e}, \mathit{uml{:}c}\rangle =$ $\{\langle x, x\rangle \mid x \in \mathit{er{:}e} \wedge \exists y.\langle y, x\rangle \in \langle \_, \mathit{uml{:}c}, \mathit{uml{:}a}\rangle\}$

This technique is particularly powerful when a data model contains **semi-structured data** which we wish to view and associate with data in a structured data model. For example, we may want to associate a URL held as an attribute in a UML model, with the web page resource in the WWW model that the URL references. In Figure 3 we illustrate how information on people in a UML model can be linked to the person's WWW home page. For this link to be established, we define an inter-model link which associates textual URLs with url constructs in the WWW model. This is achieved by the following **web page** inter-model link, which associates a resource in the WWW model to the person entity class in the UML model by the constraint that we must be able to construct the UML url attribute from the string concatenation (denoted by the '$\circ$' operator) of the uml instance in the WWW model that identifies the resource:

Higher Level Construct	
Model	inter
Construct	web page
Classification	linking, constraint
Scheme	$\langle r, c, a\rangle$

$\Rightarrow$

Equivalent HDM Representation	
Edge	$\langle \_, \mathit{www{:}r}, \mathit{uml{:}a}\rangle$
Linked-to	$\langle \mathit{www{:}r}\rangle, \langle \mathit{uml{:}a}\rangle$
Constraint	$\langle \_, \mathit{www{:}r}, \mathit{uml{:}a}\rangle = \{r, a \mid \exists w, z, s, us, pw, h, pt, up.$ $\langle w, r\rangle \in \langle \mathit{inter{:}}\mathsf{identifies}, \mathit{inter{:}}\mathsf{url}, \mathit{inter{:}}\mathsf{resource}\rangle \wedge$ $a = s \circ \text{`://'} \circ us \circ \text{`:'} \circ pw \circ \text{`@'} \circ h \circ \text{`:'} \circ pt \circ \text{`/'} \circ up \wedge$ $\langle z, a\rangle \in \langle \_, \mathit{uml{:}c}, \mathit{uml{:}a}\rangle\}$

We may use such inter-model links to write inter-model queries. For example, if we want to retrieve the WWW pages of all people that work in the computing department, we can ask the query:

$$
\begin{aligned}
\exists d,p,u.\quad & \langle d, \text{`computing'}\rangle \in \langle \mathsf{department}, \mathsf{dname}\rangle \wedge \\
& \langle d,p\rangle \in \langle \mathsf{worksin}, \mathsf{person}, \mathsf{department}\rangle \wedge \\
& \langle p,u\rangle \in \langle \mathsf{person}, \mathsf{url}\rangle \wedge \\
& \langle r,u\rangle \in inter(\langle \mathsf{resource}, \mathsf{person}, \mathsf{url}\rangle)
\end{aligned}
$$

We may also use the inter-model links to derive constructs in one model based on information held in another model. For example, we could populate the keyword attribute of person in the UML model by using a HTMLGetMeta(r,n) utility which extracts the CONTENT part of a HTML META tag in resource r, where the HTTP-EQUIV or NAME field matches n [12].

$$
\begin{aligned}
add_A^{uml}\{\quad & \mathsf{person}, \mathsf{keywords}, \{0..N\}, \{1..N\}, \\
& \{\quad \langle p,k\rangle \mid \exists u,r \,.\, \langle p,u\rangle \in \langle \mathsf{person}, \mathsf{url}\rangle \wedge \\
& \qquad \langle r,u\rangle \in inter(\langle \mathsf{resource}, \mathsf{person}, \mathsf{url}\rangle) \wedge \\
& \qquad k \in \mathrm{HTMLGetMeta}(r, \text{`Keywords'}) \\
& \} \\
\}\quad &
\end{aligned}
$$

7 Conclusions

We have presented a method for specifying semantic data models in terms of the constructs of a low-level hypergraph data model (HDM). We showed how these specifications can be used to automatically derive the transformation operations for the higher-level data models in terms of the operations of the HDM, and how these higher-level transformations can be used to perform inter-model transformations which allow data and queries to be translated from one model to another. Finally, we showed how to use the hypergraph data structure to define inter-model links, and hence allow queries which span more than one model.

Our approach clearly distinguishes between the structural and the constraint aspects of a data model. This has the practical advantage that constraint checking need only be performed when it is required to ensure consistency between models, whilst data/query access uses the structural information to translate data and queries between models.

Combined with our previous work on intra-model transformation [9, 13, 10], we have provided a complete formal framework in which to describe the semantic transformation of data and queries from almost any data source, including those containing semi-structured data. Our framework thus fits well into the various database integration architectures, such as Garlic [14] and TSIMMIS [3]. It complements these existing approaches by handling multiple data models in a more flexible manner than simply converting them all into some high level CDM such as an ER Model. It does this by representing all models in terms of their elemental nodes, edges and constraints, and allows the free mixing of different models by the definition of inter-model links. Indeed, by itself, our framework forms a useful method for the formal comparison of the semantics of various data modelling languages.

Our method has in part been implemented in a simple prototype tool. We plan now to develop a full-strength tool supporting the graphical display and manipulation of model definitions, and the definition of templates for composite transformations. We also plan to develop our approach to model dynamic aspects of conceptual modelling languages, such as dataflow diagrams, or the dynamic parts of UML.

References

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

[2] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (URL). Technical Report RFC 1738, Internet, December 1994.

[3] S.S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J.D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, October 1994.

[4] UML Consortium. UML notation guide: version 1.1. Technical report, Rational Software, September 1997.

[5] G. DeGiacomo and M. Lenzerini. A uniform framework for concept definitions in description logics. *Journal of Artificial Intelligence Research*, 6, 1997.

[6] P. Devanbu and M.A. Jones. The use of description logics in KBSE systems. *ACM Transactions on Software Engineering and Methodology*, 6(2):141–172, 1997.

[7] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, Inc., 2nd edition, 1994.

[8] S.W. Liddle, D.W. Embley, and S.N. Woodfield. Cardinality constraints in semantic data models. *Data & Knowledge Engineering*, 11(3):235–270, 1993.

[9] P.J. McBrien and A. Poulovassilis. A formal framework for ER schema transformation. In *Proceedings of ER'97*, volume 1331 of *LNCS*, pages 408–421, 1997.

[10] P.J. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — a schema transformation approach. Technical Report TR98-10, King's College London, 1998.

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

[12] C. Musciano and B. Kennedy. *HTML: The Definitive Guide.* O'Reilly & Associates, 1996.

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

[14] M.T. Roth and P. Schwarz. Don't scrap it, wrap it! A wrapper architecture for data sources. In *Proceedings of the 23rd VLDB Conference*, pages 266–275, Athens, Greece, 1997.

[15] A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990.

[16] G. Wiederhold. Forward to special issue on intelligent integration of information. *Journal on Intelligent Information Systems*, 6(2–3):93–97, 1996.