From: 'Advanced Information Systems Engineering'
11th International Conference, CAiSE'99,
M. Jarke and A. Oberweis (Eds),
Springer Verlag LNCS 1626, Pages 333--348, 1999

With minor corrections

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 aspects of the data models being integrated. Our approach is to define each of these data models in terms of a lower-level hypergraph-based data model. We show how such definitions can be used to automatically derive schema transformation operators for the higher-level data models. We also show how these higher-level transformations can be used to perform inter-model transformations, and to define inter-model links.

1 Introduction

Common to many areas of system integration is the requirement to extract data associated with a particular **universe of discourse (UoD)** represented 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)** [16] and convert all the other modelling languages into that CDM.

Using a 'higher-level' 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 because 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 — what we call the **hypergraph data model (HDM)**. We define a small set of primitive transformation operations on schemas expressed in the HDM. Higher-level modelling languages are handled by defining their constructs and transformations in terms of those of the HDM. In common with **description logics** [5, 6] we can form a union of different modelling languages to model a certain UoD. However, our approach has the advantage that it clearly separates the modelling of data structure from the modelling of constraints on the data. We note also that our HDM differs from graph-based conceptual modelling languages such as Telos [13] by supporting a very small set of low-level, elemental modelling primitives (nodes, edges and constraints). This makes the HDM better suited for use as a CDM than higher-level modelling languages, for the reasons discused in the previous paragraph.

Our previous work [14, 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 [14] we defined the notions of schemas and schema equivalence for the low-level 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 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 higher-level languages.

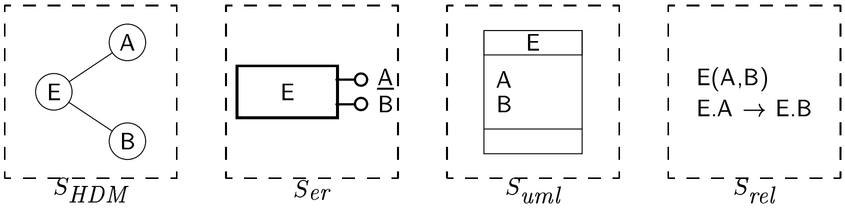

Fig. 1. Multiple models based on the HDM

Here we extend our previous work by providing a generic approach to defining the semantics of modelling languages in terms of the HDM, which in turn allows 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. In combination with the work in [10], this allows us to automatically transform queries between schemas defined in different modelling languages. Also, 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 which shows three high-level schemas each of which is represented by the same underlying HDM schema. The constructs of each of the three higher-level modelling languages (UML, ER and relational) are reduced to nodes associated by edges in the underlying HDM schema. In particular, the three schemas illustrated have a common HDM representation as a graph with three nodes and two edges, as well as some (unillustrated) constraints on the possible instances this 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 Section 5 where we demonstrate how to use our approach to handle combinations of existing modelling languages, enhanced with **inter-model links**. A summary of the paper and our conclusions are given in Section 6.

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. We refer the reader to [14, 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;
(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 (domain integrity);
(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**. A **model** is a triple $\langle S, I, Ext_{S,I}\rangle$. Two schemas are **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

[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.

S and S' are **conditionally equivalent** w.r.t f if they each conditionally subsume each other w.r.t. f. We first developed these definitions of schemas, instances, and schema equivalence in the context of an ER common data model, in earlier work [9, 11]. 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. Proviso: $toName$ is not already the name of some node.
2. $renameEdge$ $\langle\langle fromName, c_1, \ldots, c_m\rangle, toName\rangle$ renames an edge. Proviso: $toName$ is not already the name of some edge.
3. $addConstraint$ c adds a new constraint c. Proviso: c evaluates to true.
4. $delConstraint$ c deletes a constraint. Proviso: c exists.
5. $addNode$ $\langle name, q\rangle$ adds a node named $name$ whose extent is given by the value of the query q. Proviso: 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). Proviso: 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. Proviso: 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. q states how the extent of the deleted edge could be recovered from the extents of the remaining schema constructs. Proviso: the edge exists and participates in no edges.

For each of these transformations, there is a 3-ary 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. 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 equivalent. 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 conditionally equivalent w.r.t f.

It is also easy to see that every successful primitive transformation t is reversible by another successful primitive transformation t^{-1}, e.g. $addNode$ $\langle n, q\rangle$ can be reversed by $delNode$ $\langle n, q\rangle$, *etc.* This reversibility generalises to successful composite transformations, the reverse of a transformation $t_1; \ldots; t_n$ being $t_n^{-1}; \ldots; t_1^{-1}$.

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 URLs, resources and links, and showing how these too can be represented in the HDM.

In general 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. The **scheme** of each construct uniquely identifies the construct in M. For example, ER model entities may be present without requiring the presence of any other particular constructs, and their scheme is the entity name. A nodal construct maps into a node in the HDM.
- **linking** constructs can only exist in a model when certain other nodal constructs exist. 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.
- **nodal-linking** constructs are nodal constructs that can only exist when certain other nodal constructs exist, and that are linked to these constructs. Attributes in ER models are an example. Nodal-linking constructs map into a combination of a node and an edge in the HDM.

Constraint constructs represent restrictions on the extents of the extentional constructs of M. For example, ER generalisation hierarchies restrict the extent of each subclass entity to be a subset of the extent of the superclass entity, and ER 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.

Table 1 illustrates this classification of schema constructs by defining the main constructs of ER Models and giving their equivalent HDM representation. We discuss this representation in greater detail in Section 3.1 below.

The general method for constructing the set of primitive transformations for some modelling language 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 zero or one *addNode* transformations, the operand being taken from the Node field of the construct definition (if any), followed by zero or one *addEdge* transformations taken from the Edge field, followed by a sequence of zero or more *addConstraint* transformations taken from the Cons(traint) field.

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

(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.

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. For example, Table 2 shows the result of this automatic process for the ER model definition of Table 1.

Table 1. Definition of ER Model constructs

Higher Level Construct		Equivalent HDM Representation	
Construct	**entity** (E)	Node	$\langle er{:}e \rangle$
Class	nodal		
Scheme	$\langle e \rangle$		
Construct	**attribute** (A)	Node	$\langle er{:}e{:}a \rangle$
Class	nodal-linking, constraint	Edge	$\langle \_, er{:}e, er{:}e{:}a \rangle$
		Links	$\langle er{:}e \rangle$
Scheme	$\langle e, a, s_1, s_2 \rangle$	Cons	makeCard($\langle \_, er{:}e, er{:}e{:}a \rangle, s_1, s_2$)
Construct	**relationship** (R)	Edge	$\langle er{:}r, er{:}e_1, er{:}e_2 \rangle$
Class	linking, constraint	Links	$\langle er{:}e_1 \rangle, \langle er{:}e_2 \rangle$
Scheme	$\langle r, e_1, e_2, s_1, s_2 \rangle$	Cons	makeCard($\langle er{:}r, er{:}e_1, er{:}e_2 \rangle, s_1, s_2$)
Construct	**generalisation** (G)	Links	$\langle er{:}e \rangle, \langle er{:}e_1 \rangle, \ldots, \langle er{:}e_n \rangle$
Class	constraint	Cons	$er{:}e{:}g[\forall 1 \leq i < j \leq n \,.\, er{:}e_i \cap er{:}e_j = \emptyset]$; $er{:}e{:}g[\forall 1 \leq i \leq n \,.\, er{:}e_i \subseteq er{:}e]$; if $pt = \mathsf{total}$ then $er{:}e{:}g[er{:}e = \bigcup_{i=1}^{n} er{:}e_i]$
Scheme	$\langle pt, e, g, e_1, \ldots, e_n \rangle$		

3.1 An ER Model

We now look more closely at how our HDM framework can support an ER model with binary relationships and generalisation hierarchies ([14] shows how the framework can support ER models with n-ary relations, attributes on relations, and complex attributes). The representation is summarised in Table 1. We use some short-hand notation for expressing cardinality constraints on edges, in that makeCard($\langle name, c_1, \ldots, c_m \rangle, s_1, \ldots, s_m$) denotes the following constraint on the edge $\langle name, c_1, \ldots, c_m \rangle$:

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

Here, each s_i is a set of integers representing the possible values for the cardinality of the participating construct c_i e.g. $\{0, 6..10, 20..N\}$, N denoting infinity.

Table 2. Derived transformations on ER models

Transformation on er	Equivalent Transformation on HDM
$rename^{er}_{E}\langle e, e'\rangle$	$renameNode\ \langle er{:}e,\ er{:}e'\rangle$
$add^{er}_{E}\langle e, q\rangle$	$addNode\ \langle er{:}e, q\rangle$
$del^{er}_{E}\langle e, q\rangle$	$delNode\ \langle er{:}e, q\rangle$
$rename^{er}_{A}\langle a, a'\rangle$	$renameNode\ \langle er{:}e{:}a,\ er{:}e{:}a'\rangle$
$add^{er}_{A}\langle e, a, s_1, s_2, q_{att}, q_{assoc}\rangle$	$addNode\ \langle er{:}e{:}a, q_{att}\rangle$; $addEdge\ \langle\langle \_,\ er{:}e,\ er{:}e{:}a\rangle, q_{assoc}\rangle$; $addConstraint$ 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$(\langle _,\ er{:}e,\ er{:}e{:}a\rangle, s_1, s_2)$; $delEdge\ \langle\langle \_,\ er{:}e,\ er{:}e{:}a\rangle, q_{assoc}\rangle$; $delNode\ \langle er{:}e{:}a, q_{att}\rangle$
$rename^{er}_{R}\langle\langle r, e_1, e_2\rangle, r'\rangle$	$renameEdge\ \langle\langle er{:}r,\ er{:}e_1,\ er{:}e_2\rangle,\ er{:}r'\rangle$
$add^{er}_{R}\langle r, e_1, e_2, s_1, s_2, q\rangle$	$addEdge\ \langle\langle er{:}r,\ er{:}e_1,\ er{:}e_2\rangle, q\rangle$; $addConstraint$ 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$(\langle er{:}r,\ er{:}e_1,\ er{:}e_2\rangle, s_1, s_2)$; $delEdge\ \langle\langle er{:}r,\ er{:}e_1,\ er{:}e_2\rangle, q\rangle$
$rename^{er}_{G}\langle e, g, g'\rangle$	$renameConstraint\ \langle er{:}e{:}g,\ er{:}e{:}g'\rangle$
$add^{er}_{G}\langle pt, e, g, e_1, \ldots, e_n\rangle$	if $pt = \mathsf{total}$ then $addConstraint\ er{:}e{:}g[er{:}e = \bigcup_{i=1}^{n} er{:}e_i]$; $addConstraint\ er{:}e{:}g[\forall 1 \le i \le n\ .\ er{:}e_i \subseteq er{:}e]$; $addConstraint\ er{:}e{:}g[\forall 1 \le i < j \le n\ .\ er{:}e_i \cap er{:}e_j = \emptyset]$
$del^{er}_{G}\langle pt, e, g, e_1, \ldots, e_n\rangle$	if $pt = \mathsf{total}$ then $delConstraint\ er{:}e{:}g[er{:}e = \bigcup_{i=1}^{n} er{:}e_i]$; $delConstraint\ er{:}e{:}g[\forall 1 \le i < j \le n\ .\ er{:}e_i \cap er{:}e_j = \emptyset]$; $delConstraint\ er{:}e{:}g[\forall 1 \le i \le n\ .\ er{:}e_i \subseteq er{:}e]$

This notation was identified by [8] as the most expressive method for specifying cardinality constraints.

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.

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.

Relationships in ER schemas map to edges in the HDM and are as classified linking constructs. As with attributes, the cardinality constraints on relationships lead to them being classified also as constraint constructs. 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. To simplify the specification of different variants of the same transformation, we use a **conditional template transformation** of 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. Otherwise the result of the template is the identity transformation. 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'.

Table 3. Definition of relational model constructs

Higher Level Construct		Equivalent HDM Representation	
Construct	**relation** (R)	Node	$\langle rel{:}r \rangle$
Class	nodal		
Scheme	$\langle r \rangle$		
Construct	**attribute** (A)	Node	$\langle rel{:}r{:}a \rangle$
Class	nodal-linking, constraint	Edge	$\langle \_, rel{:}r, rel{:}r{:}a \rangle$
Scheme	$\langle r, a, n \rangle$	Links	$\langle rel{:}r \rangle$
		Cons	if ($n =$ null) then makeCard($\langle \_, rel{:}r, rel{:}r{:}a \rangle, \{0,1\}, \{1..N\}$) else makeCard($\langle \_, rel{:}r, rel{:}r{:}a \rangle, \{1\}, \{1..N\}$)
Construct	**primary key** (P)	Links	$\langle rel{:}r{:}a_1 \rangle, \ldots, \langle rel{:}r{:}a_n \rangle$
Class	constraint	Cons	$x \in \langle rel{:}r \rangle \leftrightarrow x = \langle x_1, \ldots, x_n \rangle$ $\wedge \langle x, x_1 \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_1 \rangle \wedge \ldots$ $\wedge \langle x, x_n \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_n \rangle$
Scheme	$\langle r, a_1, \ldots, a_n \rangle$		
Construct	**foreign key** (F)	Links	$\langle rel{:}r{:}a_1 \rangle, \ldots, \langle rel{:}r{:}a_n \rangle$
Class	constraint	Cons	$\langle x, x_1 \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_1 \rangle \wedge \ldots$ $\langle x, x_n \rangle \in \langle \_, rel{:}r, rel{:}r{:}a_n \rangle \rightarrow \langle x_1, \ldots, x_n \rangle \in rel{:}r_f$
Scheme	$\langle r, r_f, a_1, \ldots, a_n \rangle$		

3.2 The Relational Model

We define in Table 3 how the basic relational data model can be represented in the HDM. We take the relational model to consist of relations, attributes (which may be null), a primary key for each relation, and foreign keys. Our descriptions for this model, and for the following ones, omit the definitions of the primitive transformation operations since these are automatically derivable.

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 define the extent of the relation to be that of its primary key. **Attributes** in the relational model are similar to attributes of entity classes in the ER model. However, the cardinality constraint is now a simple choice between the attribute being optional (null $\rightarrow \{0,1\}$) or mandatory (notnull $\rightarrow \{1\}$). A **primary key** is a constraint that checks whether the extent of r is the same as the extents of the key attributes $a_1, \ldots, a_n$. A **foreign key** is a set of attributes $a_1, \ldots, a_n$ appearing in r that are the primary key of another relation r_f.

Table 4. Definition of UML static structure constructs

Higher Level Construct		Equivalent HDM Representation
Construct	**class** (C)	Node $\langle uml{:}c \rangle$
Class	nodal	
Scheme	$\langle c \rangle$	
Construct	**meta class** (M)	Node $\langle uml{:}m \rangle$
Class	nodal, constraint	Cons $c \in \langle uml{:}m \rangle \rightarrow \langle c \rangle$
Scheme	$\langle m \rangle$	
Construct	**attribute** (A)	Node $\langle uml{:}c{:}a \rangle$
Class	nodal-linking, constraint	Edge $\langle \_, uml{:}c, uml{:}c{:}a \rangle$
		Links $\langle uml{:}c \rangle$
Scheme	$\langle c, a, s \rangle$	Cons makeCard($\langle \_, uml{:}c, uml{:}c{:}a \rangle, s, \{1..N\}$)
Construct	**object** (O)	Links $\langle uml{:}c \rangle, \langle uml{:}c, uml{:}a_1 \rangle, \ldots, \langle uml{:}c, uml{:}a_n \rangle$
Class	constraint	Cons $c{:}o[\exists i \in uml{:}c \,.\, \langle i, v_1 \rangle \in \langle uml{:}c, uml{:}a_1 \rangle \wedge \ldots \wedge \langle i, v_n \rangle \in \langle uml{:}c, uml{:}a_n \rangle]$
Scheme	$\langle c, o, a_1, \ldots, a_n, v_1, \ldots, v_n \rangle$	
Construct	**association** ($Assoc$)	Edge $\langle uml{:}r{:}l_1{:}\ldots{:}l_n, uml{:}c_1, \ldots, uml{:}c_n \rangle$
Class	linking, constraint	Links $\langle uml{:}c_1 \rangle, \ldots, \langle uml{:}c_n \rangle$
Scheme	$\langle r, c_1, \ldots, c_n, l_1, \ldots, l_n, s_1, \ldots, s_n \rangle$	Cons makecard($\langle uml{:}r{:}l_1{:}\ldots{:}l_n, uml{:}c_1, \ldots, uml{:}c_n \rangle, s_1, \ldots, s_n$)
		Links $\langle uml{:}c \rangle, \langle uml{:}c_1 \rangle, \ldots, \langle uml{:}c_n \rangle$
Construct	**generalisation** (G)	Cons if disjoint $\in cs$ then $uml{:}c{:}g[\forall 1 \leq i < j \leq n \,.\, uml{:}c_i \cap uml{:}c_j = \emptyset]$; if complete $\in cs$ then $uml{:}c{:}g[uml{:}c = \bigcup_{i=1}^{n} uml{:}c_i]$; $uml{:}c{:}g[\forall 1 \leq i \leq n \,.\, uml{:}c_i \subseteq uml{:}c]$
Class	constraint	
Scheme	$\langle cs, c, g, c_1, \ldots, c_n \rangle$	

3.3 UML Static Structure Diagrams

We define in Table 4 those elements of UML class diagrams that model static aspects of the UoD. Elements of class diagrams that are identified as dynamic in the UML Notation Guide [4] e.g. operations, are beyond the scope of this paper.

UML **classes** are defined in a similar manner to ER entities. **Metaclasses** are defined like classes, with the additional constraint that the instances of a metaclass must themselves be classes. **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. We represent this by a single cardinality constraint, s. The cardinality constraint on the attribute is by definition $\{1..N\}$. Note that we do not restrict the domain of a in any way, so we can 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. An **object** in UML constrains the instances of some class, in the sense that the class must have an instance where the attributes $a_1, \ldots, a_n$ take specified values $v_1, \ldots, v_n$. We model this an HDM constraint, labelled with the name of the object, o, and the class, c, it is an instance of.

UML supports both binary and n-ary **associations**. Since the former is just a special case of the latter [4], 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 $s_1, \ldots, s_n$. We identify the association in the HDM by concatenating the association name with the role names. The **composition** construct is special case of an association. It has $\{1\}$ cardinality on the number of instances of the parent class that each instance of the child class is associated with (and further restrictions on the dynamic behaviour of these classes). Finally, UML **generalisations** may be either incomplete or complete, and overlapping or disjoint — giving two template transformations to handle these distinctions.

3.4 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 the 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*:identify,*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 — see Table 5 — enforcing the existence of this instance in the extension of the WWW schema.

4 Inter-Model Transformations

Our HDM representation of higher-level modelling languages is such that it is possible to unambiguously represent the constructs of multiple higher-level schemas 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.

Table 5. Definition of WWW Constructs

Higher Level Construct		Equivalent HDM Representation	
Construct	**url** (u)	Links	$\langle www{:}\mathsf{url}\rangle$
Class	constraint	Cons	$\langle s, us, pw, h, pt, up\rangle \in \langle www{:}\mathsf{url}\rangle$
Scheme	$\langle s, us, pw, h, pt, up\rangle$		
Construct	**resource** (r)	Links	$\langle www{:}\mathsf{url}\rangle$
Class	constraint	Cons	$\langle\langle s, us, pw, h, pt, up\rangle, r\rangle \in$ $\langle www{:}\mathsf{identify}, www{:}\mathsf{url}, www{:}\mathsf{resource}\rangle$
Scheme	$\langle\langle s, us, pw, h, pt, up\rangle, r\rangle$		
Construct	**link** (l)	Links	$\langle www{:}\mathsf{url}\rangle$
Class	constraint	Cons	$\langle r, \langle s, us, pw, h, pt, up\rangle\rangle \in$ $\langle www{:}\mathsf{link}, www{:}\mathsf{resource}, www{:}\mathsf{url}\rangle$
Scheme	$\langle r, \langle s, us, pw, h, pt, up\rangle\rangle$		

(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 **inter-model transformations** to be applied, 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 data and queries. This allows data and queries to be translated between different schemas in interoperating database architectures such as database federations [16] and mediators [17].

(d) New **inter-model edges** which do not belong to any single higher-level modelling language can be defined. This allows associations to be built between constructs in different modelling languages, and navigation 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.

Items (a) and (d) are discussed further in Section 5. Item (c) follows from our work in [10] which shows how schema transformations can be used to automatically migrate data and queries. We further elaborate on item (b) here. 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, $add_C^{uml}\langle\mathsf{man}, \mathsf{male}\rangle$ is equivalent to $add_C^{uml}\langle\mathsf{man}, uml(\mathsf{male})\rangle$, meaning add a UML class man whose extent is the same as that of the UML class male, while $add_C^{uml}\langle\mathsf{man}, er(\mathsf{male})\rangle$ would populate the instances of UML class man with the instances of ER entity male.

Example 1 A Relational to ER inter-model transformation
The following composite transformation transforms the relational schema S_{rel} in Figure 1 to the ER schema S_{er}. $rel(q)$ indicates that the query q should be evaluated with respect to the relational schema constructs, and $er(q)$ that q 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. $add^{er}_{A}\langle \mathsf{E}, \mathsf{A}, \mathrm{getCard}(\langle \_, rel{:}\mathsf{E}, rel{:}\mathsf{E}{:}\mathsf{A}\rangle), rel(\{y \mid \exists x.\langle x, y\rangle \in \langle \_, \mathsf{E}, \mathsf{A}\rangle\}), rel(\langle \_, \mathsf{E}, \mathsf{A}\rangle)\rangle$
3. $add^{er}_{A}\langle \mathsf{E}, \mathsf{B}, \mathrm{getCard}(\langle \_, rel{:}\mathsf{E}, rel{:}\mathsf{E}{:}\mathsf{B}\rangle), rel(\{y \mid \exists x.\langle x, y\rangle \in \langle \_, \mathsf{E}, \mathsf{B}\rangle\}), rel(\langle \_, \mathsf{E}, \mathsf{B}\rangle)\rangle$
4. $del^{rel}_{P}\langle \mathsf{E}, \mathsf{A}\rangle$
5. $del^{rel}_{A}\langle \mathsf{E}, \mathsf{B}, \mathrm{getCard}(\langle \_, er{:}\mathsf{E}, er{:}\mathsf{E}{:}\mathsf{B}\rangle), er(\{y \mid \exists x.\langle x, y\rangle \in \langle \_, \mathsf{E}, \mathsf{B}\rangle\}), er(\langle \_, \mathsf{E}, \mathsf{B}\rangle)\rangle$
6. $del^{rel}_{A}\langle \mathsf{E}, \mathsf{A}, \mathrm{getCard}(\langle \_, er{:}\mathsf{E}, er{:}\mathsf{E}{:}\mathsf{A}\rangle), er(\{y \mid \exists x.\langle x, y\rangle \in \langle \_, \mathsf{E}, \mathsf{A}\rangle\}), er(\langle \_, \mathsf{E}, \mathsf{A}\rangle)\rangle$
7. $del^{rel}_{R}\langle \mathsf{E}, er(\{y \mid \exists x.\langle x, y\rangle \in \langle \_, \mathsf{E}, \mathsf{A}\rangle\})\rangle$

Notice that the reverse transformation from S_{er} back to S_{rel} is automatically derivable from this transformation, as discussed at the end of Section 2, and consists of a sequence of transformations $add^{rel}_{R}, add^{rel}_{A}, add^{rel}_{A}, add^{rel}_{P}, del^{er}_{A}, del^{er}_{A}$, del^{er}_{E} whose arguments are the same as those of their counterparts in the forward direction.

Whilst it is possible to write inter-model transformations such as this one 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 followed in preparing these template transformations:

1. Ensure that every possible instance of a 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 below. However, usually it is combinations of instances of constructs in M_1 that map to instances of constructs in M_2, as the remaining transformation steps in Example 2 illustrate.
2. Ensure that every construct c of M_1 appears in a transformation that deletes c, recovering the extent of c from the extents of constructs in M_2 that were created in the addition transformations during Step 1 above.

We illustrate Step 1 in Example 2 by showing how the constructs of any relational model can be mapped to constructs of an ER model.

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 the instances of the 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 an 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 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 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 primary key attribute of the ER class. Based on this principle, we could define the new construct *common class* shown in Table 6.

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 2 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 **web page** inter-model link in Table 6, 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 url instance in the WWW model that identifies the resource.

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:

$\exists d, p, u\, .\, \langle d, \text{'computing'}\rangle \in \langle \mathsf{dept}, \mathsf{dname}\rangle \wedge \langle d, p\rangle \in \langle \mathsf{worksin}, \mathsf{person}, \mathsf{dept}\rangle \ \wedge$
$\langle p, u\rangle \in \langle \mathsf{person}, \mathsf{url}\rangle \wedge \langle r, u\rangle \in inter(\langle \mathsf{resource}, \mathsf{person}, \mathsf{url}\rangle)$

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 keywords attribute of person in the UML model by using the 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].

Table 6. Two Examples of Inter-model Constructs

Higher Level Construct		Equivalent HDM Representation	
Construct	**common class**	Edge	$\langle \_, er{:}e, uml{:}c \rangle$
Class	linking, constraint	Links	$\langle er{:}e \rangle, \langle uml{:}c \rangle$
Scheme	$\langle e, c, a \rangle$	Cons	$\langle \_, er{:}e, uml{:}c \rangle = \{\langle x, x \rangle \mid x \in er{:}e \wedge \exists y . \langle y, x \rangle \in \langle \_, uml{:}c, uml{:}a \rangle\}$
Construct	**web page**	Edge	$\langle \_, www{:}r, uml{:}a \rangle$
Class	linking, constraint	Links	$\langle www{:}r \rangle, \langle uml{:}a \rangle$
Scheme	$\langle r, c, a \rangle$	Cons	$\langle \_, www{:}r, uml{:}a \rangle = \{\langle r, a \rangle \mid \exists z, s, us, pw, h, pt, up. \langle\langle s, us, pw, h, pt, up \rangle, r \rangle \in \langle www{:}\mathsf{identity}, www{:}\mathsf{url}, www{:}\mathsf{resource} \rangle \wedge a = s \circ \text{'://'} \circ us \circ \text{':'} \circ pw \circ \text{'@'} \circ h \circ \text{':'} \circ pt \circ \text{'/'} \circ up \wedge \langle z, a \rangle \in \langle \_, uml{:}c, uml{:}a \rangle\}$

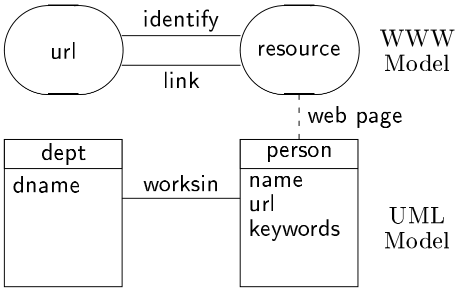

Fig. 2. Linking a semantic data model with the WWW

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

6 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. Finally, we showed how to use the hypergraph data structure to define inter-model links, hence allowing queries which span multiple models.

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 can use the structural information to translate data and queries between models.

Combined with our previous work on intra-model transformation [9, 14, 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 [15] 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 extend our approach to model dynamic aspects of conceptual modelling languages and to support temporal data models.

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. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4):325–362, October 1990.

14. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
15. 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.
16. A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990.
17. G. Wiederhold. Forward to special issue on intelligent integration of information. *Journal on Intelligent Information Systems*, 6(2–3):93–97, 1996.