From: 'Advanced Information Systems Engineering',
14th International Conference, CAiSE 2002,
A. Banks Pidduck *et al* (Eds),
Springer Verlag LNCS 2348, Pages 484-499, 2002

Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach

Peter McBrien[1] and Alexandra Poulovassilis[2]

[1] Dept. of Computing, Imperial College,
180 Queen's Gate, London SW7 2BZ, pjm@doc.ic.ac.uk
[2] Dept. of Computer Science, Birkbeck College, University of London,
Malet Street, London WC1E 7HX, ap@dcs.bbk.ac.uk

Abstract. This paper presents a new approach to schema evolution, which combines the activities of schema integration and schema evolution into one framework. In previous work we have developed a general framework to support schema transformation and integration in heterogeneous database architectures. Here we show how this framework also readily supports *evolution of source schemas*, allowing the global schema and the query translation pathways to be easily repaired, as opposed to having to be regenerated, after changes to source schemas.

1 Introduction

Common to many methods for integrating heterogeneous data sources is the requirement for *logical integration* [21, 9] of the data, due to variations in schema design for the same universe of discourse. Logical integration requires facilities for transforming and integrating a set of source schemas into a global schema, and for translating queries posed on the global schema to queries over the source schemas. In previous work [14, 15, 19, 17, 16] we have developed a general framework which provides these facilities. In this paper we consider the problem of evolving the global schema, and repairing the query translation pathways, as source schemas evolve.

Other heterogeneous database systems, such as TSIMMIS [6], InterViso [22], IM [13], and Garlic [20], are what may be termed *query-oriented.* They provide mechanisms by which users define global schema constructs as views over source schema constructs (or vice versa in the case of IM). More recent work on automatic wrapper generation [23, 7, 2, 8] and agent-based mediation [3] is also query-oriented. In contrast, our approach is *schema transformation-oriented.* We provide a flexible framework by which transformations on schemas can be specified. These transformations are then used to automate the translation of queries between global and source schemas. Clio [18] also fits into this category, using the specification of correspondences between constructs as a basis for translating data. However it is limited in the range of transformations it can perform.

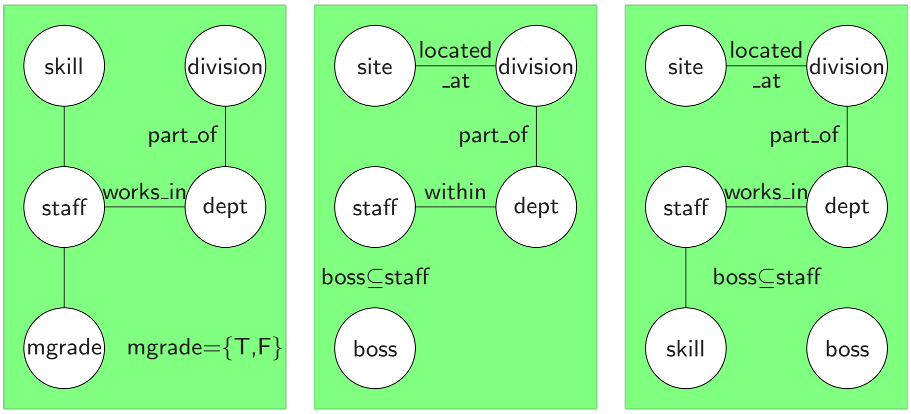
Our transformation-oriented approach provides a set of primitive schema transformations each of which makes a 'delta' change to the schema, adding, deleting or renaming just one schema construct. These primitive transformations are incrementally composed into more complex schema transformations. As we will see below, an advantage of our approach over query-oriented approaches is that it allows systematic repair of global schemas as source schemas evolve.

Much of the previous work on schema evolution has presented approaches in terms of just one data model e.g. [1, 4, 5, 11]. In contrast, we represent higher-level data modelling languages in terms of an underlying hypergraph-based data model [17]. Thus, the techniques for handling source schema evolutions that we propose in this paper can be applied to any of the common data modelling languages. In [12] it was argued that a uniform approach to schema evolution and schema integration is both desirable and possible, and this is our view also. The higher-order logic language *SchemaLog* was used to define the relationship between schemas, contrasting with our approach which uses a simple set of schema transformation primitives augmented with a first-order query language. A particular advantage of our approach is that we clearly distinguish between equivalent and non-equivalent constructs in different schemas, and hence are able to distinguish between queries that can and cannot be translated between the two schemas. This ability to specify *capacity-augmentations* is also present in the approach of [4], but that work is specific to object-oriented schemas and not readily transferable to other data models.

The remainder of this paper is as follows. Section 2 reviews the hypergraph data model that underpins our approach. Section 3 shows how global schemas and global query translation can be repaired in the face of source schema evolution, considering in particular the evolution of a source schema into a semantically equivalent, semantically contracted, or semantically expanded schema. Section 4 shows how the same approach can be used to repair global schemas defined using higher-level modelling languages. Section 5 gives our conclusions.

2 Review of Our Framework

A **schema** in the **hypergraph data model** (HDM) is a triple $\langle Nodes, Edges, Constraints\rangle$. $Nodes$ and $Edges$ define a labelled, directed, nested hypergraph. It is 'nested' in the sense that edges may link any number of both nodes and other edges (necessary in order to support higher-level constructs such as composite attributes and attributes on relations [19]). It is a directed hypergraph because edges link sequences of nodes or edges. A **query** q **over a schema** $S = \langle Nodes, Edges, Constraints\rangle$ is an expression whose variables are members of $Nodes \cup Edges$. $Constraints$ is a set of boolean-valued queries over S. Nodes have unique names. Edges and constraints have an optional name associated with them. The nodes and edges of a schema are identified by their **scheme**. For a node this is of the form $\langle\langle nodeName\rangle\rangle$ and for an edge it is of the form $\langle\langle edgeName, scheme_1, scheme_2, \ldots, scheme_n\rangle\rangle$.



(a) S_1: staff skills (b) S_2: staff locations (c) S: global schema

Fig. 1. Two HDM source schemas, and a global schema

Figure 1 illustrates three HDM schemas that we will use as a running example in this paper. Schema S_1 contains information about staff, whether or not they are of manager grade, the skills of each member of staff, and the department in which they work. Each department belongs to a division. The node mgrade is constrained to have a two-valued extent $\{\mathsf{T}, \mathsf{F}\}$. Instances of $\langle\langle$staff$\rangle\rangle$ who are managers are linked to T by an instance of $\langle\langle$_,staff,mgrade$\rangle\rangle$ while other staff are linked to F. (The underscore means the edge has no name).

Schema S_2 is drawn from the same domain, and contains information about staff, their departments, and the division that each department belongs to. However, S_2 differs from S_1 in several ways: the edge between $\langle\langle$staff$\rangle\rangle$ and $\langle\langle$dept$\rangle\rangle$ has a different name in the two schemas, although its 'real-world' semantics are the same; S_2 does not contain information about staff skills; S_2 contains information about the site that each division is located at; S_2 represents the information about managers in a different manner to S_1, using a node $\langle\langle$boss$\rangle\rangle$ to hold instances of staff which are managers, together with a constraint that states that all instances of manager are also instances of staff.

Schema S is an integration of S_1 and S_2, with the edge within of S_2 having been renamed to works_in, and the node boss of S_2 being used to record information about which staff members are managers, rather than the node mgrade of S_1. We will shortly see how S_1 and S_2 are integrated into S using our framework.

The HDM is equipped with a set of **primitive transformations** on schemas. Three primitive transformations are available for adding a node n, edge e, or constraint c to an HDM schema, S: addNode(n, q), addEdge(e, q), and addCons(c). Here, q is a query on S which defines the extent of a new node or edge in terms of the extents of the existing constructs of S (so adding this new construct does not change the information content of the schema). Similarly, the following primitive transformations delete a node, edge or constraint: $\mathsf{delNode}(n, q)$, $\mathsf{delEdge}(e, q)$, and $\mathsf{delCons}(c)$. Here, q is a query on S which defines how the extent of the deleted construct can be reconstructed from the extents of the remaining schema constructs (so deleting this construct does not change the information content of the schema). There are also two primitive transformations for renaming a node or an edge: $\mathsf{renNode}(\langle\langle n\rangle\rangle, \langle\langle n^{new}\rangle\rangle)$ and $\mathsf{renEdge}(\langle\langle n, s_1, \ldots, s_m\rangle\rangle, \langle\langle n^{new}\rangle\rangle)$ (where $s_1, \ldots, s_m$ are the schemes linked by edge n).

In [16] we defined four more low-level transformations to also allow transformations between semantically overlapping schemas rather than just between semantically equivalent schemas: $\mathsf{extendNode}(n)$ is equivalent to $\mathsf{addNode}(n, void)$, where the *void* query component indicates that the new node cannot be derived from the existing schema constructs; $\mathsf{extendEdge}(e)$ is similarly equivalent to $\mathsf{addEdge}(e, void)$; $\mathsf{contractNode}(n)$ is equivalent to $\mathsf{delNode}(n, void)$, where the *void* query component indicates that the removed node cannot be derived from the remaining schema constructs; and $\mathsf{contractEdge}(e)$ is similarly equivalent to $\mathsf{delEdge}(e, void)$.

Example 1 Building the global schema S. Schemas S_1 and S_2 are transformed into S by the two composite transformations listed below.

In step 1, the query $\{x \mid x \in \langle\langle \mathsf{staff}\rangle\rangle; \langle x, \mathsf{T}\rangle \in \langle\langle \_, \mathsf{staff}, \mathsf{mgrade}\rangle\rangle\}$ states how to populate the extent of the new node $\langle\langle \mathsf{boss}\rangle\rangle$ from the extents of the existing schema constructs (indicating that $\langle\langle \mathsf{boss}\rangle\rangle$ adds no new information to the schema). In particular, the extent of $\langle\langle \mathsf{boss}\rangle\rangle$ is those instances of $\langle\langle \mathsf{staff}\rangle\rangle$ linked to the value T of $\langle\langle \mathsf{mgrade}\rangle\rangle$ by an instance of the $\langle\langle \_, \mathsf{staff}, \mathsf{mgrade}\rangle\rangle$ edge.

In 3, the query $\{x, \mathsf{T} \mid x \in \langle\langle \mathsf{boss}\rangle\rangle\} \cup \{x, \mathsf{F} \mid x \in \langle\langle \mathsf{staff}\rangle\rangle - \langle\langle \mathsf{boss}\rangle\rangle\}$ states how the extent of the deleted edge $\langle\langle \_, \mathsf{staff}, \mathsf{mgrade}\rangle\rangle$ can be recovered from the remaining schema constructs (indicating that this edge is a redundant construct). In particular, an instance $\langle x, \mathsf{T}\rangle$ is created for each manager x and an instance $\langle x, \mathsf{F}\rangle$ is created for each staff member x who is not a manager.

In 5, the query $\{\mathsf{T}, \mathsf{F}\}$ states how the extent of the deleted node mgrade can be recovered, in this case by a simple enumeration of its two values.

The rest of the steps are straight-forward:

transformation $S_1 \rightarrow S$
1 *addNode* $\langle\langle \mathsf{boss}\rangle\rangle$ $\{x \mid \langle x, \mathsf{T}\rangle \in \langle\langle \_, \mathsf{staff}, \mathsf{mgrade}\rangle\rangle\}$
2 *addCons* $\langle\langle \mathsf{boss}\rangle\rangle \subseteq \langle\langle \mathsf{staff}\rangle\rangle$
3 *delEdge* $\langle\langle \_, \mathsf{staff}, \mathsf{mgrade}\rangle\rangle$ $\{x, \mathsf{T} \mid x \in \langle\langle \mathsf{boss}\rangle\rangle\} \cup \{x, \mathsf{F} \mid x \in \langle\langle \mathsf{staff}\rangle\rangle - \langle\langle \mathsf{boss}\rangle\rangle\}$
4 *delCons* $\langle\langle \mathsf{mgrade}\rangle\rangle = \{\mathsf{T}, \mathsf{F}\}$
5 *delNode* $\langle\langle \mathsf{mgrade}\rangle\rangle$ $\{\mathsf{T}, \mathsf{F}\}$
6 *extendNode* $\langle\langle \mathsf{site}\rangle\rangle$
7 *extendEdge* $\langle\langle \mathsf{located\_at}, \mathsf{division}, \mathsf{site}\rangle\rangle$

transformation $S_2 \rightarrow S$
8 *renEdge* $\langle\langle \mathsf{within}, \mathsf{staff}, \mathsf{dept}\rangle\rangle$ works_in
9 *extendNode* $\langle\langle \mathsf{skill}\rangle\rangle$
10 *extendEdge* $\langle\langle \_, \mathsf{staff}, \mathsf{skill}\rangle\rangle$

Schema transformations defined on the HDM, or on higher-level modelling languages defined in terms of the HDM, are *automatically reversible* [16]. In particular, every add transformation step is reversible by a del transformation with the same arguments. For example, the reverse transformations from S to S_1 and from S to S_2 are automatically generated from the two transformations given in Example 1 and are as follows, where each step $\overline{t}$ is the reverse of step t in Example 1:

transformation $S \rightarrow S_1$
$\overline{7}$ *contractEdge* ⟨⟨located_at,division,site⟩⟩
$\overline{6}$ *contractNode* ⟨⟨site⟩⟩
$\overline{5}$ *addNode* ⟨⟨mgrade⟩⟩ $\{\mathsf{T}, \mathsf{F}\}$
$\overline{4}$ *addCons* ⟨⟨mgrade⟩⟩$= \{\mathsf{T}, \mathsf{F}\}$
$\overline{3}$ *addEdge* ⟨⟨_,staff,mgrade⟩⟩ $\{x, \mathsf{T} \mid x \in \langle\langle \mathsf{boss} \rangle\rangle\} \cup \{x, \mathsf{F} \mid x \in \langle\langle \mathsf{staff} \rangle\rangle - \langle\langle \mathsf{boss} \rangle\rangle\}$
$\overline{2}$ *delCons* ⟨⟨boss⟩⟩ $\subseteq$ ⟨⟨staff⟩⟩
$\overline{1}$ *delNode* ⟨⟨boss⟩⟩ $\{x \mid \langle x, \mathsf{T} \rangle \in \langle\langle \_,\mathsf{staff},\mathsf{mgrade} \rangle\rangle\}$

transformation $S \rightarrow S_2$
$\overline{10}$ *contractEdge* ⟨⟨_,staff,skill⟩⟩
$\overline{9}$ *contractNode* ⟨⟨skill⟩⟩
$\overline{8}$ *renEdge* ⟨⟨works_in,staff,dept⟩⟩ within

In [16] we show how this reversibility of schema transformations allows automatic query translation between schemas. In particular, if a schema S is transformed to a schema S' by a single primitive transformation step, the only cases that need to be considered in order to translate a query Q posed on S to a query Q' posed on S' are ren transformations, in which case the renaming needs to be applied in reverse, and del transformations, in which case occurrences in Q of the deleted construct need to be substituted by the query q specified in the transformation. For sequences of primitive transformations, these substitutions are successively applied in order to obtain the final translated query Q'.

This translation scheme can be applied to each of the constructs of a global schema in order to obtain the possible derivations of each construct from the set of source schemas. These derivations can then be substituted into any query over the global schema in order to obtain an equivalent query over the source schemas [10]. To illustrate, consider the following query on schema S, which finds the skills of staff members working within divisions based at the London site:

$\{sk \mid \langle s, sk \rangle \in \langle\langle \_,\mathsf{staff},\mathsf{skill} \rangle\rangle;\ \langle s, dep \rangle \in \langle\langle \mathsf{works\_in},\mathsf{staff},\mathsf{dept} \rangle\rangle;$
$\quad \langle dep, div \rangle \in \langle\langle \mathsf{part\_of},\mathsf{dept},\mathsf{division} \rangle\rangle;$
$\quad \langle div, \text{‘London’} \rangle \in \langle\langle \mathsf{located\_at},\mathsf{division},\mathsf{site} \rangle\rangle\}$

Applying our translation scheme to this query, gives the following query over the constructs of S_1 and S_2, where we distinguish the constructs of these schemas by the suffixing them by 1 or 2, respectively:

$\{sk \mid \langle s, sk \rangle \in \langle\langle \_,\mathsf{staff},\mathsf{skill} \rangle\rangle_1 \cup void;$
$\quad \langle s, dep \rangle \in \langle\langle \mathsf{works\_in},\mathsf{staff},\mathsf{dept} \rangle\rangle_1 \cup \langle\langle \mathsf{within},\mathsf{staff},\mathsf{dept} \rangle\rangle_2;$
$\quad \langle dep, div \rangle \in \langle\langle \mathsf{part\_of},\mathsf{dept},\mathsf{division} \rangle\rangle_1 \cup \langle\langle \mathsf{part\_of},\mathsf{dept},\mathsf{division} \rangle\rangle_2;$
$\quad \langle div, \text{‘London’} \rangle \in void \cup \langle\langle \mathsf{located\_at},\mathsf{division},\mathsf{site} \rangle\rangle_2\}$

The first line of the query indicates that $\langle s, sk\rangle$ may only be retrieved from the ⟨⟨_,staff,skill⟩⟩ edge in S_1, the second line indicates that $\langle s, dep\rangle$ may be retrieved from either ⟨⟨works_in,staff,dept⟩⟩ in S_1 or from ⟨⟨within,staff,dept⟩⟩ in S_2, the third line that $\langle dep, div\rangle$ may be retrieved from ⟨⟨part_of,dept,division⟩⟩ in S_1 or S_2, and the last line that $\langle div, \text{'London'}\rangle$ may be retrieved from ⟨⟨located_at,division,site⟩⟩ in S_2. Standard optimisation techniques can now be applied to this translated query, in order to generate a reasonable query plan for execution.

3 Handling Evolution of Source Schemas

We turn now to the main theme of this paper, namely how global schemas can be repaired (as opposed to regenerated) in order to reflect changes in source schemas, and how query translation operates over the repaired global schema. Although our examples assume HDM schemas and queries/constraints expressed in a comprehension language, the treatment is fully general and applies to higher-level schema constructs and other query formalisms. We discuss this issue further in Section 4, by illustrating the approach applied to UML models.

Let us suppose then that there are n source schemas S_1, ..., S_n which have been transformed and integrated into a global schema S. There are thus available n transformations $T_1 : S_1 \rightarrow S$, ..., $T_n : S_n \rightarrow S$. From these, the reverse transformations $\overline{T_1} : S \rightarrow S_1$, ..., $\overline{T_n} : S \rightarrow S_n$ are automatically generated and can be used to translate queries posed on S to queries on $S_1, ..., S_n$.

The source schema evolution problem that we consider is illustrated in Figure 2 and is as follows: if some source schema S_i evolves, to S'_i say, how should S be repaired to reflect this change and how should queries on the repaired S now be translated in order to operate on S'_i rather than on S_i?

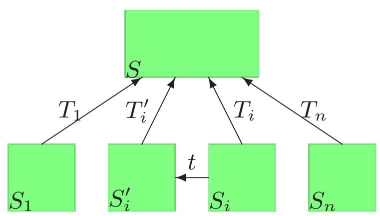


Fig. 2. Evolution of a source schema S_i

The first step is to specify the change from S_i to S'_i by a transformation. The second step is to analyse this transformation. Without loss of generality, we need only consider changes on S_i that consist of a *single* primitive transformation step t being applied to give S'_i, since changes that are composite transformations can be handled as a sequence of primitive transformations. A new transformation pathway T'_i from S'_i to S can be automatically generated as $T'_i = \overline{t}; T_i$ (and so

$\overline{T_i'}$ from S to S_i' is $\overline{T_i'} = \overline{T_i}; t$). How we now handle T_i' depends on what type of transformation t was:

1. If t was an add, delete or rename transformation, then the new schema S_i' will be equivalent S_i. In this case, all information in the global schema S that was derivable from S_i may now be derived from S_i', and no changes are required to S. Section 3.1 illustrates such an equivalence preserving transformation for our example. The only issue is that T_i' might be simplifiable, which we consider in Section 3.2.
2. If t was a contract transformation, then there will be some information which used to be present in S_i that will no longer be available from S_i'. In Section 3.3 we discuss the circumstances in which this will require S to also be contracted.
3. If t was an expand transformation, then the relationship of the new construct with what is already present in S needs to be investigated, and we discuss this in Section 3.4.

A final point to note is that this treatment also covers the case where a new schema S_{n+1} is added to the set of source schemas from which S is derived, in that the original schema S_{n+1} can be regarded as an empty schema which is successively expanded with new constructs. Similarly, our treatment covers the removal of a source schema S_i, since its constructs can be successively contracted to leave an empty schema.

3.1 Equivalence-preserving transformations

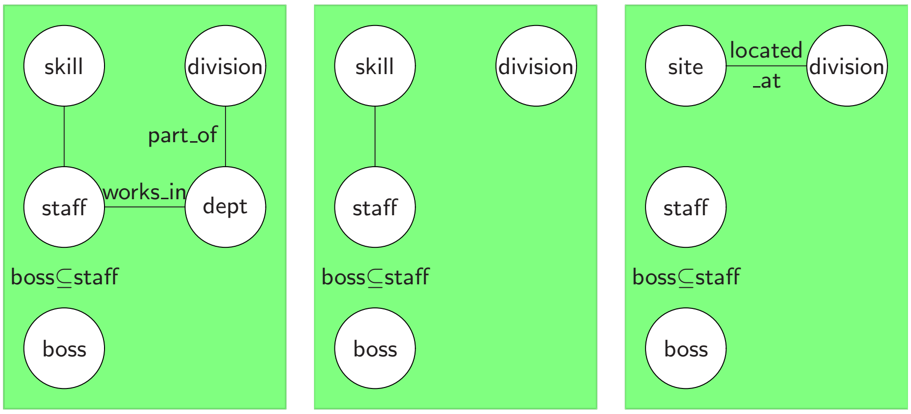
Suppose that t is an equivalence-preserving transformation, so that S_i' is equivalent to S_i. Then T_i' and $\overline{T_i'}$ as defined above provide a new automatic translation pathway between S and the new source schema S_i'.

Example 2

Suppose that it has been decided to evolve schema S_1 in Figure 1 to a new equivalent schema S_1^a which models the notion of a manager in the same way as S_2 (see Figure 3(a)). This can be achieved by the following composite transformation:

transformation $S_1 \rightarrow S_1^a$
⑪ *addNode* $\langle\langle \mathsf{boss} \rangle\rangle$ $\{x \mid \langle x, \mathsf{T} \rangle \in \langle\langle \_,\mathsf{staff},\mathsf{mgrade} \rangle\rangle\}$
⑫ *addCons* $\langle\langle \mathsf{boss} \rangle\rangle \subseteq \langle\langle \mathsf{staff} \rangle\rangle$
⑬ *delEdge* $\langle\langle \_,\mathsf{staff},\mathsf{mgrade} \rangle\rangle$ $\{x, \mathsf{T} \mid x \in \langle\langle \mathsf{boss} \rangle\rangle\} \cup \{x, \mathsf{F} \mid x \in \langle\langle \mathsf{staff} \rangle\rangle - \langle\langle \mathsf{boss} \rangle\rangle\}$
⑭ *delCons* $\langle\langle \mathsf{mgrade} \rangle\rangle = \{\mathsf{T}, \mathsf{F}\}$
⑮ *delNode* $\langle\langle \mathsf{mgrade} \rangle\rangle$ $\{\mathsf{T}, \mathsf{F}\}$

The reverse transformation, ⑮, ⑭, ⑬, ⑫, ⑪, from S_1^a to S_1 is automatically generated. This is prefixed to the transformation $S_1 \rightarrow S$ of Example 1 to give the new transformation from S_1^a to S:



(a) S_1^a (b) S_1^b (c) S_2^a

Fig. 3. Evolution of source schemas

transformation $S_1^a \rightarrow S$
15 *addNode* ⟨⟨mgrade⟩⟩ {T, F}
14 *addCons* ⟨⟨mgrade⟩⟩= {T, F}
13 *addEdge* ⟨⟨_,staff,mgrade⟩⟩ $\{x, \mathsf{T} \mid x \in$ ⟨⟨boss⟩⟩$\} \cup \{x, \mathsf{F} \mid x \in$ ⟨⟨staff⟩⟩ $-$ ⟨⟨boss⟩⟩$\}$
12 *delCons* ⟨⟨boss⟩⟩ ⊆ ⟨⟨staff⟩⟩
11 *delNode* ⟨⟨boss⟩⟩ $\{x \mid \langle x, \mathsf{T}\rangle \in$ ⟨⟨_,staff,mgrade⟩⟩$\}$
1 *addNode* ⟨⟨boss⟩⟩ $\{x \mid \langle x, \mathsf{T}\rangle \in$ ⟨⟨_,staff,mgrade⟩⟩$\}$
2 *addCons* ⟨⟨boss⟩⟩ ⊆ ⟨⟨staff⟩⟩
3 *delEdge* ⟨⟨_,staff,mgrade⟩⟩ $\{x, \mathsf{T} \mid x \in$ ⟨⟨boss⟩⟩$\} \cup \{x, \mathsf{F} \mid x \in$ ⟨⟨staff⟩⟩ $-$ ⟨⟨boss⟩⟩$\}$
4 *delCons* ⟨⟨mgrade⟩⟩= {T, F}
5 *delNode* ⟨⟨mgrade⟩⟩ {T, F}
6 *extendNode* ⟨⟨site⟩⟩
7 *extendEdge* ⟨⟨located_at,division,site⟩⟩

Conversely, the new transformation from S to S_1^a is automatically obtained by appending steps 11–15 to the transformation $S \rightarrow S_1$ of Example 1. This new transformation can now be used to automatically translate queries posed on S to queries on S_1^a rather than on S_1.

3.2 Removing redundant transformation steps

In composite transformations such as those above there may be pairs of primitive transformation steps which are inverses of each other and which can be removed without altering the overall effect of the transformation. In particular, a composite transformation $T; t; T'; \bar{t}; T''$, where T, T', T'' are arbitrary sequences of primitive transformations, t is primitive transformation and $\bar{t}$ is its inverse, can be simplified to $T; T'; T''$ provided that there are no references within T' to the construct being renamed, added or deleted by t. For example, in the transformation $S_1^a \rightarrow S$, steps 11 and 1 can be removed, followed by 12 and 2, 13 and 3, 14 and 4, and 15 and 5, obtaining the following expected simplified transformation:

transformation $S_1^a \rightarrow S$
6 *extendNode* ⟪site⟫
7 *extendEdge* ⟪located_at,division,site⟫

Renaming transformations may also be redundant, and hence removable. For example, $ren\ c\ c'; del\ c' \equiv del\ c$, since there is no point in renaming c only to delete it. Similarly $add\ c'; ren\ c'\ c \equiv add\ c$ and $ren\ c'\ c''; ren\ c''\ c \equiv ren\ c'\ c$.

3.3 Contraction transformations

Suppose S_i is transformed to S_i' by a primitive transformation t of the form *contract* c. The new transformation pathway from S to S_i' is $\overline{T_i}$; *contract* c. Any sub-queries over S that translate to the construct c of S_i will now correctly be replaced by the value *void* over S_i'.

However, after a series of contractions on source schemas, the global schema S may eventually contain constructs that are no longer supported by *any* source schema. How can S be repaired so that it no longer contains such unsupported constructs? One way is by *dynamic repair* during query processing: if a sub-query posed on a construct of S returns *void* for all possible local sub-queries, then that construct can be removed from S. Note that if this construct participates in any edges in S, then these must be removed first (if the construct has already been removed from all the source schemas, then so must any edges that it participated in there, and so such edges will also be redundant in S).

Another way to repair S if it contains constructs that are no longer supported by any source schema is by *static repair*. With this approach, we can first use T_i to trace how the removed construct c of S_i is represented in S — call this global representation $global(c)$. The transformations $\overline{T_j}$ $j \neq i$ can be used to trace how $global(c)$ is represented in all the other source schemas S_j, $j \neq i$. If all of these source constructs have *void* extents, then $global(c)$ can be removed from S (again taking care to precede the removal of a construct by removal of any edges that it participates in).

Example 3 illustrates how the removal of a construct from one source schema may or may not still allow the construct to be derived from another source schema.

Example 3 Contractions of source schemas.
Suppose the owner of schema S_1^a has decided not to export information about departments. Schema S_1^b illustrated in Figure 3(b) is derived from S_1^a as follows:

transformation $S_1^a \rightarrow S_1^b$
⑯ *contractEdge* ⟨⟨works_in,staff,dept⟩⟩
⑰ *contractEdge* ⟨⟨part_of,dept,division⟩⟩
⑱ *contractNode* ⟨⟨dept⟩⟩

Adopting a static repair approach to repairing S involves checking the transformation paths of the contracted constructs dept, works_in and part_of from S to S_1^b and S_2, and would discover that all of them still map to a non-*void* extent in S_2. Thus, S would not be changed. With a dynamic repair approach, it would be found that queries on S over dept, works_in or part_of can still be posed on S_2 and so S would again not be changed.

Suppose now that S_2 is also transformed in a similar manner, resulting in S_2^a illustrated in Figure 3(c):

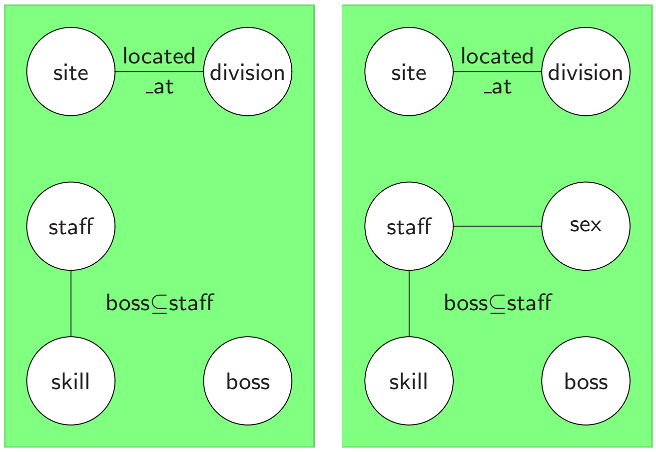
transformation $S_2 \rightarrow S_2^a$
⑲ *contractEdge* ⟨⟨within,staff,dept⟩⟩
⑳ *contractEdge* ⟨⟨part_of,dept,division⟩⟩
㉑ *contractNode* ⟨⟨dept⟩⟩

At this stage the transformations from S to S_1^b and S_2^a are as follows:

transformation $S \rightarrow S_1^b$
$\overline{7}$ *contractEdge* ⟨⟨located_at,division,site⟩⟩
$\overline{6}$ *contractNode* ⟨⟨site⟩⟩
⑯ *contractEdge* ⟨⟨works_in,staff,dept⟩⟩
⑰ *contractEdge* ⟨⟨part_of,dept,division⟩⟩
⑱ *contractNode* ⟨⟨dept⟩⟩

transformation $S \rightarrow S_2^a$
$\overline{10}$ *contractEdge* ⟨⟨_,staff,skill⟩⟩
$\overline{9}$ *contractNode* ⟨⟨skill⟩⟩
$\overline{8}$ *renEdge* ⟨⟨works_in,staff,dept⟩⟩ within
⑲ *contractEdge* ⟨⟨within,staff,dept⟩⟩
⑳ *contractEdge* ⟨⟨part_of,dept,division⟩⟩
㉑ *contractNode* ⟨⟨dept⟩⟩

Adopting a static repair approach again means checking the transformation paths of the contracted constructs dept, works_in and part_of from S to S_1^b and S_2^a. In this case all three of them map to a *void* extent in both source schemas. Thus, they are removed from S, obtaining the schema S^a illustrated in Figure 4. The corresponding *contract* steps are also removed, as are any prior renamings of these constructs, from the transformations from S^a to the source schemas S_1^b and S_2^a (in the reverse transformations the corresponding *extend* steps would be removed). With a dynamic repair approach, it would be found that queries over dept, works_in and part_of on S translate to *void* on all source schemas, and the same actions would be taken. With both approaches, the resulting transformation from S^a to S_1^b is $\overline{7}$, $\overline{6}$ and from S^a to S_2^a is $\overline{10}$, $\overline{9}$.



(a) S^a (b) S^b

Fig. 4. Global schema following evolution of source schemas

We note that it is not actually *wrong* in our framework for a construct from a global schema to map to a *void* extent in all source schemas — for example, a new source schema may later be added that does support an extent for this construct, and this is likely to be a common situation in mediator architectures. Sub-queries over such constructs would merely translate to *void*.

3.4 Extension transformations

Suppose S_i is transformed to S'_i by a primitive transformation t of the form *extend* c, meaning that a new construct c is now supported by S'_i that is not derivable from S_i. Naively, the new transformation pathway from S to S'_i is $\overline{T_i}$; *extend* c. However, this may be incorrect and there are four alternatives to be considered:

1. c does not appear in S but can be derived from S by some transformation T. In this case, S is transformed to a new global schema S' that contains c by appending T to the transformation from each local schema to the original S. The transformation pathway from S' to S'_i then simplifies to just $\overline{T_i}$ i.e. $\overline{T}$ and *extend* c are inverses of each other and can be removed.
2. c does not appear in S, and cannot be derived from S.
   In this case, S is transformed to a new global schema S' that contains c by appending the step *extend* c to the transformation from each local schema to the original S. The reverse transformation from S' to S'_i thus consists of an initial *contract* c step. This matches up with the newly appended *extend* c

step. This pair of steps must be removed in order for the new extent of c in S'_i to be usable by queries posed on S'.

3. c appears in S and has the same semantics as the newly added c in S'_i.
In this case there must be a transformation step *contract* c in the original transformation from S to S_i. This matches up with the newly appended *extend* c in the transformation from S to S'_i. This pair of steps must be removed in order for the new extent of c in S'_i to be usable by queries posed on S.
4. c appears in S but has different semantics to the newly added c in S'_i.
In this case there must again be a transformation step *contract* c in the original transformation from S to S_i. Now, the new construct c in S'_i needs to be renamed to some name that does not appear in S, c', say. The resulting transformation from S to S'_i is $\overline{T_i}; extend\ c'; ren\ c'c$, and the situation reverts to case 2 above.

In 1 to 4 above, determining whether the new construct c can or cannot be derived from the existing constructs of S requires domain knowledge (as does specifying the transformation T in 1) e.g. from a human expert or a domain ontology. After this is determined, the repair steps on S and the transformation pathways can be performed automatically.

By analogy to our remark at the end of Section 3.3 (that it is not compulsory to repair the global schema after a series of contractions have left a global schema construct unsupported by any source schema), it is similarly not compulsory to extend the global schema after a new construct is added to a source schema in cases 2 and 4 above. If this is the choice, then the *extend* c step is not appended to the transformations from the source schemas to the global schema, and the final *extend* c step remains in the transformation from S to S'_i.

Example 4 Extensions of source schemas.

Suppose that the owner of S_2^a (Figure 3(c)) has decided to extend it into a new schema S_2^b containing information about staff members' skills and their sex. This can be achieved by the following transformation:

transformation $S_2^a \rightarrow S_2^b$
22 *extendNode* ⟨⟨skill⟩⟩
23 *extendEdge* ⟨⟨_,staff,skill⟩⟩
24 *extendNode* ⟨⟨sex⟩⟩
25 *extendEdge* ⟨⟨_,staff,sex⟩⟩

Comparing S_2^b with S^a (Figure 4(a)), it is apparent that the constructs introduced by 22 and 23 already appear in S^a. For these two constructs a choice must be made between cases 3 and 4 above. Let us suppose that both constructs have the same semantics in S^a and S_2^b, so that case 3 holds. Examining the transformations $S^a \rightarrow S_2^a$ and $S_2^a \rightarrow S_2^b$, the redundant pair 9 and 22 can be eliminated, as can the redundant pair 10 and 23. This results in the following transformation from S^a to S_2^b:

transformation $S^a \rightarrow S_2^b$
(24) *extendNode* ⟨⟨sex⟩⟩
(25) *extendEdge* ⟨⟨_,staff,sex⟩⟩

Suppose now that the constructs introduced by (24) and (25) are entirely new ones, so that case 2 above applies. Then S^a is extended to S^b using these same two extend steps. The two redundant contract/extend pairs are then removed from the transformation $S^b \rightarrow S_2^b$, giving the expected identity transformation from S_2^b and S^b.

4 Handling Higher-Level Modelling Languages

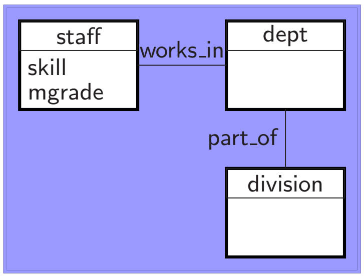
In [17] we showed how higher-level modelling languages L can be expressed using the HDM, representing each construct c in L as a set of constructs $c_1, \ldots, c_n$ in the HDM. We then showed how *add*, *del* and *ren* primitive transformations on c can be automatically derived as sequences of primitive HDM transformations on $c_1, \ldots, c_n$. In particular, we showed how primitive transformations for UML schemas such as *addClass, delClass, addAttribute, delAttribute, addAssociation, delAssociation, addGeneralisation, delGeneralisation*, are defined in terms of the set of primitive transformations on HDM schemas presented in Section 2. Thus, the schema evolution methodology presented in Section 3 transfers directly to this higher semantic level.

To illustrate this, Figure 5(a) shows a UML class diagram U_1 which is semantically equivalent to the HDM schema S_1 of Figure 1(a). The nodes ⟨⟨staff⟩⟩, ⟨⟨dept⟩⟩ and ⟨⟨division⟩⟩ are represented as classes, and ⟨⟨mgrade⟩⟩ and ⟨⟨skill⟩⟩ as attributes of staff. The ⟨⟨works_in,staff,dept⟩⟩ and ⟨⟨part_of,dept,division⟩⟩ edges are represented as associations. Figure 5(d) shows a UML class diagram U which is semantically equivalent to the HDM schema S of Figure 1(c). The node ⟨⟨boss⟩⟩ from S is represented as a class in U, and the constraint boss $\subseteq$ staff is represented by the generalisation hierarchy between staff and boss. Transforming U_1 to U may be achieved by the following steps:

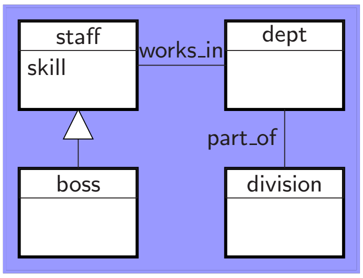
transformation $U_1 \rightarrow U$
(U1) *addClass* boss $\{x \mid x \in \mathsf{staff}; \langle x, \mathsf{T}\rangle \in \mathsf{staff.mgrade}\}$
(U2) *addGeneralisation* (staff,boss)
(U3) *delAttribute* staff.mgrade $\{x, \mathsf{T} \mid x \in \mathsf{boss}\} \cup \{x, \mathsf{F} \mid x \in \mathsf{staff}; x \notin \mathsf{boss}\}$
(U4) *extendAttribute* division.site

Note that each of the steps in $U_1 \rightarrow U$ equates with one or more of the steps in $S_1 \rightarrow S$. In particular, (U1) adding UML class boss is equivalent to (1), (U2) adding the UML generalisation is equivalent to (2), (U3) deleting a UML attribute is equivalent to (3) and (4), and (U4) extending the UML schema with an attribute is equivalent to (6) and (7).

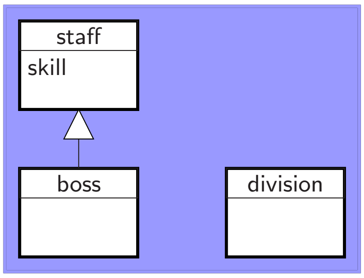
Figure 5(b) shows a schema U_1^a which is semantically equivalent to S_1^a. The following is an equivalence-preserving transformation from U_1 to U_1^a (again, the steps in this transformation can be equated with those in $S_1 \rightarrow S_1^a$):



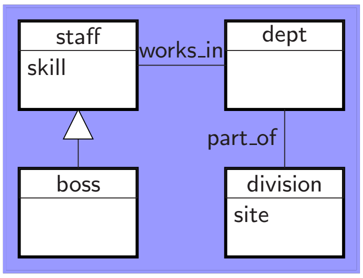
(a) U_1: UML version of S_1



(b) U_1^a: UML version of S_1^a



(c) U_1^b: UML version of S_1^b



(d) U: UML version of S

Fig. 5. Transformations on UML class diagrams

transformation $U_1 \rightarrow U_1^a$
U5 *addClass* boss $\{x \mid x \in \mathsf{staff}; \langle x, \mathsf{T} \rangle \in \mathsf{staff.mgrade}\}$
U6 *addGeneralisation* (staff,boss)
U7 *delAttribute* staff.mgrade $\{x, \mathsf{T} \mid x \in \mathsf{boss}\} \cup \{x, \mathsf{F} \mid x \in \mathsf{staff}; x \notin \mathsf{boss}\}$

Since all UML schema transformations are equivalent to some HDM schema transformation, our analysis from Section 3 can be applied to the evolution of source schemas expressed in UML. For example, since $U_1 \rightarrow U_1^a$ is an equivalence-preserving transformation, then the steps of $U_1 \rightarrow U_1^a$ allow any query on U that used to execute on U_1 to instead execute on U_1^a using the transformation pathway $U \rightarrow U_1; U_1 \rightarrow U_1^a$ = U4, U3, U2, U1, U5, U6, U7. Applying the removal of redundant pairs of transformations in Section 3.2 to the higher level of UML simplifies $U \rightarrow U_1^a$ to just U4.

Our analysis of contraction and extension of HDM source schemas can also be applied at the UML level in the obvious way. For example, the following UML contraction from U_1^a to U_1^b is equivalent to the HDM contraction $S_1^a \rightarrow S_1^b$:

transformation $U_1^a \rightarrow U_1^b$
U8 *contractAssociation* works_in

U9 *contractAssociation* part_of
U10 *contractClass* dept

Thus, after U_1^a has evolved to U_1^b, the same analysis as in Section 3.3 applies, where dept, works_in and part_of in U will map to *void* in U_1^b, but still map to an non-*void* extent in U_2 (the unillustrated UML equivalent of S_2). The remaining examples of Sections 3.3 and 3.4 transfer to this UML level in a similar way.

5 Concluding Remarks

In this paper we have shown how our framework for schema transformation provides a uniform approach to handling both schema integration and schema evolution in heterogeneous database architectures. Source schemas are integrated into a global schema by applying a sequence of primitive transformations to them. The same set of primitive transformations can be used to specify the evolution of a source schema into a new schema. We have shown how the transformations between the source schemas and the global schema can be used to systematically repair the global schema and the query translation pathways as source schemas evolve, considering in particular the evolution of a source schema into a semantically equivalent, semantically contracted, or semantically expanded schema. The first two cases can be handled totally automatically, and the third case semi-automatically.

Our framework is based on a low-level hypergraph-based data model whose primitive constructs are nodes, edges, and constraints. In previous work we have shown how this HDM supports the representation and transformation of a wide variety of higher-level data modelling languages. Our use of the relatively simple HDM means that our approach to schema evolution is straightforward to analyse while at the same time being applicable to real-world modelling situations using more complex data models for database schemas. We are currently implementing the schema transformation, integration and evolution functionality described here within the AutoMed project (`http://www.doc.ic.ac.uk/automed/`).

References

1. J. Andany, M. Leonard, and C. Palisser. Management of schema evolution in databases. In *Proceedings of VLDB'91*. Morgan-Kaufman, 1991.
2. N. Ashish and C.A. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, 26(4):8–15, December 1997.
3. R.J. Bayardo *et al.* InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. *SIGMOD Record*, 26(2):195–206, June 1997.
4. Z. Bellahsene. View mechanism for schema evolution in object-oriented dbms. In *Advances in Databases: 14th British National Conference on Databases, BNCOD14*, volume 1094, pages 18–35. Springer-Verlag, 1996.
5. B. Benatallah. A unified framework for supporting dynamic schema evolution in object databases. In *Proceedings of ER99*, LNCS, pages 16–30. Springer-Verlag, 1999.

6. 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.
7. T. Critchlow, M. Ganesh, and R. Musick. Automatic generation of warehouse mediators using an ontology engine. In *Proceedings of the 5th International Workshop on Knowledge Represenation Meets Databases (KRDB '98)*, volume 10. CEUR Workshop Proceedings, 1998.
8. J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system. *SIGMOD Record*, 26(2):532–535, June 1997.
9. R. Hull. Managing sematic heterogeneity in databases: A theoretical perspective. In *Proceedings of PODS*, 1997.
10. E. Jasper. Query translation in heterogeneous database environments. MSc thesis, Birkbeck College, September 2001.
11. M. Kradolfer and A. Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *Proceedings of CoopIS 1999*, pages 104–114, 1999.
12. L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Proceedings of DOOD'93*, pages 81–100, Phoenix, AZ, December 1993.
13. A. Levy, A. Rajamaran, and J.Ordille. Querying heterogeneous information sources using source description. In *Proc 22nd VLDB*, pages 252–262, 1996.
14. 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.
15. P.J. McBrien and A. Poulovassilis. A formalisation of semantic schema integration. *Information Systems*, 23(5):307–334, 1998.
16. P.J. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — a schema transformation approach. In *Proceedings of ER99*, volume 1728 of *LNCS*, pages 96–113. Springer-Verlag, 1999.
17. P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Advanced Information Systems Engineering, 11th International Conference CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer-Verlag, 1999.
18. R.J. Miller, M.A. Hernández, L.M. Haas, L.-L. Yan, C.T.H. Ho, R. Fagin, and L. Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.
19. A. Poulovassilis and P.J. McBrien. A general formal framework for schema transformation. *Data and Knowledge Engineering*, 28(1):47–71, 1998.
20. 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.
21. A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990.
22. M. Templeton, H.Henley, E.Maros, and D.J. Van Buer. InterViso: Dealing with the complexity of federated database access. *The VLDB Journal*, 4(2):287–317, April 1995.
23. M.E. Vidal, L. Raschid, and J-R. Gruser. A meta-wrapper for scaling up to multiple autonomous distributed information sources. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS98)*, pages 148–157. IEEE-CS Press, 1998.