View Generation and Optimisation in the AutoMed Data Integration Framework

Edgar Jasper[1], Nerissa Tong[2], Peter McBrien[2], and Alexandra Poulovassilis[1]

[1] School of Computer Science and Information Systems, Birkbeck College
[2] Department of Computing, Imperial College

Abstract. In AutoMed, data integration is based on the use of reversible sequences of schema transformations. We discuss how views can be generated from these sequences. We also discuss techniques for optimising the views, firstly by simplifying the transformation sequences, and secondly by optimising the view definitions generated from them.

1 Introduction

Up to now, most data integration approaches have been either **global as view (GAV)** or **local as view (LAV)** (see [2]). One disadvantage of GAV is that it does not readily support the evolution of the local schemas. LAV isolates changes to local schemas to impact only on the derivation rules defined for that schema. However, LAV has problems if one needs to change the global schema, since all the rules for defining local schemas as views of the global schema will need to be reviewed. In [3], we presented a data integration framework based on reversible sequences of schema transformations, called transformation **pathways**. We term our approach **both as view (BAV)** since from these pathways it is possible to extract both GAV and LAV views. Moreover, BAV allows the evolution of both global and local schemas, since pathways can be incrementally modified.

We begin this short paper with a brief review of BAV in Section 2. In Section 3 we discuss the generation of GAV and LAV views. It could be argued that BAV pathways are likely to be more costly to reason with and process than the corresponding LAV or GAV view definitions. However, in Section 2 we discuss how BAV pathways are amenable to considerable simplification. Moreover, standard query optimisation techniques can be applied to the view definitions derived from them, and we discuss these in Section 3. We refer the reader to the full technical report [1] for more details and examples.

2 The BAV Integration Approach

We have implemented the BAV approach within the AutoMed system, more details of which can be found at `http://www.doc.ic.uk/automed`. AutoMed supports as its common data model a **hypergraph-based data model (HDM)** and a set of primitive schema transformations defined for this model. Higher-level data models and primitive schema transformations for them are defined in terms of this lower-level common data model.

Schemas are incrementally transformed by applying to them a sequence of primitive transformations $t_1, \ldots, t_r$. Each primitive transformation t_i makes a 'delta' change to the schema by adding, deleting or renaming just one schema construct. Each add or delete transformation is accompanied by a query specifying the extent of the new or deleted construct in terms of the rest of the constructs in the schema. This query is expressed in a functional **intermediate query language**, **IQL**.

In order to integrate n local schemas, $LS_1, \ldots, LS_n$, each LS_i first needs to be transformed into a "union-compatible" schema US_i. These $US_1, \ldots, US_n$ are syntactically identical, and this is asserted by creating a sequence of id transformation steps between each pair US_i and US_{i+1}, of the form id $(US_i\!:\!c, US_{i+1}\!:\!c)$ for each schema construct (id is an additional type of primitive transformation, and the notation $US_i\!:\!c$ distinguishes each schema's c construct). These id transformations are generated automatically by the AutoMed software. An arbitrary one of the US_i can then be selected for further transformation into a global schema GS. This is where constructs sourced from different local schemas can be combined together by unions, joins, outer-joins etc.

There may be information within a US_i which is not semantically derivable from the corresponding LS_i. This is asserted by means of extend transformation steps within the pathway $LS_i \rightarrow US_i$. Conversely, not all of the information within a local schema LS_i need be transferred into US_i and this is asserted by means of contract transformation steps within $LS_i \rightarrow US_i$. extend and contract transformations behave in the same way as add and delete, respectively, except that they indicate that their accompanying query may only partially construct the extent of the new/removed construct. Moreover, their query may just be the constant Void, indicating that the extent of the new/removed construct cannot be derived even partially.

Each primitive transformation t has an **automatically derivable** reverse transformation $\bar{t}$. In particular, each add or extend transformation is reversed by a delete or contract transformation with the same arguments, and vice versa, while each rename or id transformation is reversed by another rename or id transformation with the two arguments swapped. This holds for the primitive transformations of **any** modelling language defined in AutoMed.

We have developed a **Transformation Manipulation Language (TML)** to support analysis and simplification of BAV pathways. TML formalises a primitive transformation step t_i transforming a schema S to a schema S' as having four conditions $a_i^+, b_i^-, c_i^+, d_i^-$. The positive precondition a_i^+ is the set of constructs that t_i implies must be present in S. The negative precondition b_i^- is the set of constructs that t_i implies must not be present in S. The positive postcondition c_i^+ is the set of constructs that t_i implies must be present in S'. The negative postcondition d_i^- is the set of constructs that t_i implies must not be present in S'.

A pathway T is said to be **well-formed** if for each step t_i in T: $S' = (S \cup c_i^+) - d_i^-$, $S = (S' \cup a_i^+) - b_i^-$, $a_i^+ \subseteq S$, $b_i^- \cap S = \emptyset$, $c_i^+ \subseteq S'$ and $d_i^- \cap S' = \emptyset$.

Transformations may be swapped within T provided T remains well-formed. In particular, a pair of consecutive transformations t_i,t_{i+1} may be swapped to t_{i+1},t_i provided (a) t_i does not add a construct required by t_{i+1}, and $\overline{t_{i+1}}$ does not add a construct required by $\overline{t_i}$, *i.e.* : $(c_i^+ - a_i^+) \cap a_{i+1}^+ = \emptyset$ and $(a_{i+1}^+ - c_{i+1}^+) \cap c_i^+ = \emptyset$; and (b) t_i does not delete a construct required not to be present by t_{i+1}, and $\overline{t_{i+1}}$ does not delete a construct required not to be present by $\overline{t_i}$, *i.e.* $d_i^- \cap b_{i+1}^- = \emptyset$.

Two transformations t_x and t_y in a well-formed pathway T are **redundant** (resp. **partially redundant**) if T can be reordered such that t_x and t_y become consecutive within it and T remains well-formed if they are then removed (resp. replaced by a single transformation t_{xy}). The table below summarises the simplifications that can be applied to T when a pair of such transformations is found to operate on the same construct c. NWF denotes 'not well-formed' and [] denotes removal of the pair. The table also applies to extend and contract transformation steps by replacing add by extend, and delete by contract.

			t_y	
		add(c, q)	delete(c, q)	rename(c, c')
t_x	add(c, q)	NWF	[]	add(c', q)
	delete(c, q)	[]	NWF	NWF
	rename(c', c)	NWF	delete(c', q)	[]
	rename(c'', c)	NWF	delete(c'', q)	rename(c'', c')

3 Generating and Optimising Views

One of the strengths of BAV integration is that all primitive transformations are automatically reversible. Thus, for any two schemas linked by a primitive transformations, there exists a pathway between them and it does not matter in which direction the pathway was originally created. Thus, from the one set of pathways linking a set of schemas, we can derive both GAV and LAV views.

For GAV views, the pathways from the global schema GS to each local schema LS_i are retrieved from AutoMed's metadata repository. View definitions for each global schema construct are then derived incrementally by traversing the tree. Initially, each construct's view definition is just the construct itself. Each node in the tree is then visited in a downwards direction. The only transformations that need to be considered are those that delete, contract or rename an extensional construct. These are relevant because the current view definitions may query extensional constructs that no longer exist after such a transformation. In particular, if the transformation is a delete or a contract, occurrences in the current view definitions of the deleted construct are substituted by the query specified in the transformation. If the transformation is a rename occurrences of the renamed construct are replaced by its former name.

When the tree branches, constructs of the parent schema at the branching point are semantically identical to constructs with the same scheme within the child schemas. The possibility of using any path is retained within the view definitions by replacing each construct of the parent schema by a disjunction (OR) of the corresponding constructs in the child schemas. Child schema constructs may later be made Void by contract transformations occurring further down the tree. The tree is traversed in this fashion from the root to the leaves until all the nodes are visited. The resulting view definitions are the GAV definitions for the global schema constructs over the local schemas.

For LAV views, the pathway from a local schema to the global schema is processed as above to derive the view definitions. The process is simpler than for GAV views because there is now only a single pathway being traversed, with no branching. More generally, views can be derived for any schema in terms of any set of other schemas provided that pathways linking all the schemas exist.

View definitions can be simplified after they have been generated. This saves later work for the query optimiser when these definitions are substituted into global queries for GAV query processing (which is what AutoMed supports).

The AutoMed query language, IQL, has bags (multi-sets) as its basic collection type, and supports two kinds of operator for manipulating them: bag union, ++, and also a family of operators all derived from one function fold. fold applies a given function to each element of a bag and then 'folds' a binary operator into the resulting values. For example, sum = fold (id) (+) 0 and count = fold (lambda x.1) (+) 0. Selection, projection, join and group-by operations can also be defined in terms of fold. Optimisations for fold apply to all operators that can be defined in terms of it. Two particular optimisations can be applied to view definitions generated from BAV pathways. Firstly, instances of Void can be removed. Void is equivalent to the empty bag and thus:

```
fold f op e Void    = fold f op e []      = e
Void ++ e = [] ++ e = e ++ [] = e ++ Void = e
Void OR e           = e OR Void           = e
```

Secondly, due to the step-wise fashion in which view definitions are generated, **loop fusion** may also be applicable. This optimisation replaces two successive iterations over a collection by one iteration provided the iterations satisfy certain algebraic properties. We refer the reader to [4] for a discussion of IQL, and for references to relevant work on fold-based functional query languages and optimisation techniques for them.

References

1. E. Jasper, N. Tong, P. McBrien, and A. Poulovassilis. View generation and optimisation in the AutoMed data integration framework. Technical report, AutoMed Project, 2003.
2. M. Lenzerini. Data integration: A theorectical perspective. In *Proc. PODS02*, pages 247–258, 2002.
3. P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, 2003.
4. A. Poulovassilis. The AutoMed Intermediate Query Language. Technical report, AutoMed Project, 2001.