Schema Merging Based on Semantic Mappings

Nikos Rizopoulos and Peter McBrien

Dept. of Computing, Imperial College London, London SW7 2AZ
nr600@doc.ic.ac.uk, pjm@doc.ic.ac.uk

Abstract. In model management, the Merge operator takes as input a pair of schemas, together with a set of mappings between their objects, and returns an integrated schema. In this paper we present a new approach to implementing the Merge operator based on ***semantic mappings*** between objects. Our approach improves upon previous work by (1) using formal low-level transformation rules that can be translated into higher-level rules and (2) specifying precise BAV mappings, which merge schemas without any information loss or gain.

1 Introduction

Using the terminology of **model management** [2], the process of **data integration** requires the use of three operators. Firstly, ModelGen is used to translate schemas into some common data modelling language. Secondly, a Match operator is applied to two schemas and produces a set of mappings (sometimes called correspondences) between the objects found in the two schemas. Thirdly, the Merge operator takes this set of mappings and performs transformations on the schemas to produce a new integrated schema. In this paper, we propose an implementation of Merge based on **semantic mappings**, such as equivalence, subsumption, disjointness, *etc*, between objects.

In our approach, we have exhaustively identified all possible semantic mappings between pairs of objects, and defined formal Merge rules for all possible cases. Our rules merge schemas without any information loss or gain, and they form a framework for merging schemas represented in high level modelling languages, such as ER or relational [10]. Additionally, our fine-grained analysis identifies the cases where Merge can be fully automated and the cases which might require the user's intervention depending on the application setting.

2 The Merge Operator

Using a standard data integration approach [1], our Merge process is divided into three phases: (a) naming conforming, (b) unioning and (c) restructuring, with every phase having a set of formally defined rules. Here, we give an example of a rule for each phase.

Our approach uses a low-level graph language as the common modelling language, called the HDM [9], which represents data as *Nodes*,*Edges* and *Constraints*. For the purposes of this paper, we only use the subsumption constraint $\subseteq$, however our approach makes use of all the primitive constraint constructs in [4].

A.P. Sexton (Ed.): BNCOD 2009, LNCS 5588, pp. 193–198, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The transformation language we use in Merge is **both-as-view** (**BAV**) [9,8]. BAV is made up of primitive transformations, series of which form **pathways** which define mappings between schemas. There is the add transformation, together with its inverse delete transformation, which have the semantics that the object added/deleted is fully derivable from the other objects. In the case of nodes and edges, this means there must be a query q supplied, using a list comprehension [6] language called IQL [7], that gives the extent of the node or edge derived from the extent of nodes and edges already in the schema. Use of these transformations therefore has the **intentional domain preservation property** (**IDPP**) — there is no information gained or lost by their use. Another primitive transformation is rename, which also satisfies the IDPP. Finally, the extend transformation adds a schema object that is only partially derivable from the existing schema (i.e, query q returns a subset of the extent of the object). Use of this transformation does not satisfy the IDPP, and we wish to avoid its use in the rules defining our Merge operator.

2.1 Naming Conforming, Unioning and Restructuring

In the **naming conforming** phase, naming conflicts between the objects of the schemas are resolved. Given that there may be both homonym and synonym [1] conflicts between nodes and between edges, this leads us to have four rules: **Node Merge**, **Edge Merge**, **Node Distinction** and **Edge Distinction** rules. The Node Merge and Edge Merge rules resolve the synonym conflict, of having two equivalent ($\stackrel{S}{=}$) nodes or two equivalent edges, that do not have identical names. In such a case, the rules assign to the two equivalent objects a common name by generating two rename transformations. Formally, the Node Merge rule is the following:

$$\frac{\begin{array}{l}\mathsf{hdm{:}node}{:}\langle\langle \mathsf{N}_1 \rangle\rangle \stackrel{S}{=} \mathsf{hdm{:}node}{:}\langle\langle \mathsf{N}_2 \rangle\rangle \\ \neg\ \mathsf{identicalNames}(\mathsf{N}_1, \mathsf{N}_2) \\ \mathsf{commonName}(\mathsf{N}_1, \mathsf{N}_2, \mathsf{N}')\end{array}}{\begin{array}{l}\mathsf{rename}(\mathsf{hdm{:}node}{:}\langle\langle \mathsf{N}_1 \rangle\rangle, \mathsf{hdm{:}node}{:}\langle\langle \mathsf{N}' \rangle\rangle) \\ \mathsf{rename}(\mathsf{hdm{:}node}{:}\langle\langle \mathsf{N}_2 \rangle\rangle, \mathsf{hdm{:}node}{:}\langle\langle \mathsf{N}' \rangle\rangle)\end{array}}$$

All our rules take the form of conditions, defined above the horizontal line, which if satisfied cause the BAV transformations, below the line, to be generated.

The Node Distinction and Edge Distinction rules resolve the homonym conflict of having two non-equivalent objects with identical names. In this case, the objects have to be assigned distinct names to make them distinguishable.

The **unioning** phase is superimposing the two schemas; thus each pair of equivalent nodes, now with identical names, collapses into a single node. The superimposition of schemas in BAV is performed by a series of extend transformations. Note that this is the only place where our Merge operator uses extend, since it is stating here what *is not* directly available from one schema, but should only be sourced from the other.

In the rest of the unioning phase, subsumption, intersection and disjointness mappings between nodes are examined. The set of rules of this phase is the **Addition of Inclusion**, **Addition of Intersection** and the **Addition of Union** rules. The purpose of these rules is to identify any possible concepts that do not appear explicitly in the schemas but are implicitly defined. In addition, any appropriate constraints between the

nodes are added. For example the Addition of Inclusion rule adds the $\subseteq$ constraint for each subsumption ($\overset{S}{\subset}$) mapping:

$$\frac{\text{hdm:node:}\langle\langle N_2\rangle\rangle \overset{S}{\subset} \text{hdm:node:}\langle\langle N_1\rangle\rangle}{\text{add(hdm:con:}\langle\langle \subseteq, \text{hdm:node:}\langle\langle N_2\rangle\rangle, \text{hdm:node:}\langle\langle N_1\rangle\rangle\rangle\rangle)}$$

The Addition of Intersection rule says that if an intersection mapping holds between two nodes, the common sub-domain is explicitly represented by an added *intersection node*. The Addition of Union rule adds the *union node* of two disjoint nodes.

In the final **restructuring** phase of Merge, the existence of equivalence, subsumption, intersection and disjointness mappings between edges are examined. The set of formally defined rules of this phase includes **Redundant Edge Removal** rules, **Optional Edge Removal, Specialisation of Edges, Addition of Intersection Edge** and **Addition of Union Edge** rules. The purpose of these rules is to minimize duplication and simplify the schema. For example, the Redundant Edge Removal can be applied when there are two equivalent edges and thus one is redundant:

$$\frac{\begin{array}{l}\text{hdm:edge:}\langle\langle e_1, N_1, N'_{1/2}\rangle\rangle \overset{S}{=} \text{hdm:edge:}\langle\langle e_2, N_2, N'_{1/2}\rangle\rangle \\ \text{hdm:node:}\langle\langle N_2\rangle\rangle \overset{S}{\subset} \text{hdm:node:}\langle\langle N_1\rangle\rangle \\ \text{constraints}(N_1,e_1,\text{Cons})\end{array}}{\begin{array}{l}\text{genDeleteCons(Cons)} \\ \text{moveDependents}(e_1,e_2) \\ \text{delete(hdm:edge:}\langle\langle e_1, N_1, N'_{1/2}\rangle\rangle, \text{hdm:edge:}\langle\langle e_2, N_2, N'_{1/2}\rangle\rangle)\end{array}}$$

The predicate constraints(N_1,e_1,Cons) instantiates the variable Cons with the list of constraint objects between N_1 and e_1. The rule calls the predicate genDeleteCons(Cons) which deletes Cons. moveDependents(e_1,e_2) replace the references to e_1 with references to e_2 by generating a series of add and delete transformations.

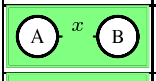
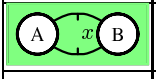
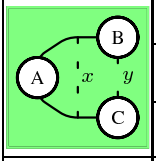
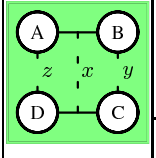
3 Merge properties

Our Merge process is based on formally defined rules. As seen in Table 1, we have exhaustively examined all possible cases of semantic mappings between objects. The first row of the table is concerned with the mapping x between nodes. The rest of the rows of the table are concerned with the mapping x between two edges.

The thorough examination of all possible mappings between two objects has identified cases where no simplification is possible, illustrated as NSP cells in the table, thus no rules have been defined for these cases. For example, edge disjointness cases can only be simplified if the nodes in at least one end of the edges are disjoint. This condition is necessary in order for the Addition of Union Edge rules to satisfy the IDPP. Additionally, we have identified cases where the semantic mappings specified between the objects are not possible, illustrated as SRNP cells in the table. For example, if the nodes at one end of the edges are disjoint then the edges cannot be equivalent based on the definition of equivalence [10].

The most important property of our rules is the IDPP. The fine-grained granularity of the BAV transformations and the low-level constraints used in the proposed approach allow us to reason about the correctness of the rules and demonstrate that the rules do

Table 1. Summary of merge rules possible for all combinations of the semantic mappings

case	y	z	x: $\subseteq$	x: $\subset$	x: $\cap$	x: $\emptyset$
			Node Merge/Distinction	Addition of Inclusion	Addition of Intersection	Addition of Union
			Edge Merge/Distinction	NSP	Addition of Edge Intersection	NSP
	$\subset$		Redundant Edge Removal	Optional Edge Removal	Addition of Edge Intersection	NSP
	$\supset$		Specialisation of Edges	NSP	Addition of Edge Intersection	NSP
	$\emptyset$		SRNP	SRNP	SRNP	Addition of Union Edge
	$\subset$	$\subset$	Redundant Edge Removal	Optional Edge Removal	Addition of Edge Intersection	NSP
		$\supset$	Specialisation of Edges	Optional Edge Removal	Addition of Edge Intersection	NSP
		$\emptyset$	SRNP	SRNP	SRNP	Addition of Union Edge
	$\supset$	$\subset$	Redundant Edge Removal	NSP	Addition of Edge Intersection	NSP
		$\supset$	Specialisation of Edges	Optional Edge Removal	Addition of Edge Intersection	NSP
		$\emptyset$	Specialisation of Edges	NSP	Addition of Edge Intersection	NSP
	$\emptyset$	$\subset$	SRNP	SRNP	SRNP	Addition of Union Edge
		$\supset$	SRNP	SRNP	SRNP	Addition of Union Edge
		$\emptyset$	SRNP	SRNP	SRNP	Addition of Union Edge

not violate the IDPP. For example, the Redundant Edge Removal rule does not violate the IDPP since the extent of the deleted edge e_1 can be retrieved from the equivalent edge e_2, as specified by the query attached to the delete transformation.

A by-product of the IDPP in the application of the rules, is that the order of their application is insignificant. Even if, for example, an object which is deleted by a rule R_1 is necessary for the application of another rule R_2, the IDPP ensures that the domain of the deleted object can still be retrieved from the current schema and therefore R_2 can still be applied. A recursive query re-writing process replaces each appearance of the deleted object in R_2 with the query specified in the object's delete transformation.

Finally, having exhaustively identified the possible rules, we can examine which rules and more specifically which parts of the rules can be fully automated and which parts need user intervention. The interesting predicates are the naming predicates used when a new object is added onto the schema. These are: commonName/3, distinctNames/4 and uniqueName/1. Possible automatic implementations of commonName/3 are: (a) identifying the common substring of the objects' names and (b) using the name of the object that belongs to the user's preferred schema. Predicate distinctNames/4 can also

be automated quite simply: the distinct names can be produced by simply prefixing the name of the objects with the name of the schema they belong to. Similarly, the uniqueName/1 predicate can be fully automated by identifying a random name or by concatenating the names of the objects the rule is dealing with. In some cases, such implementations might be sufficient, *e.g.* in a meta-search engine where the merged schema is not presented to the user. On the other hand, if the schema is available to the user or it is used in some other data integration scenario, such a fully automated implementation might be problematic.

4 Summary and Conclusions

In this paper, we have presented an overview of our generic and formal framework to the Merge operator, which is used to produce a single integrated schema based on two existing schemas and supplied semantic mappings between the schema objects.

Our Merge process is based on formal rules, whose soundness and completeness can be proved, and whose fine-grained granularity enables the identification of the steps that can either be fully or semi-automated. Our approach is defined as rules over a low-level modelling language, the HDM, and it can form a framework for high-level model schema merging as illustrated in [10].

The work most related to ours is [3], where a generic Merge operator is proposed based on work in [5]. Two types of correspondences, equivalence and similarity, between nodes can be specified, but Merge cannot interpret the general similarity correspondence. Correspondences between edges cannot be specified. Instead, our approach examines both nodes and edges and in addition deals with a much wider set of mappings between them. However, in [3] multiple-type conflicts are resolved, which we do not consider. Cardinality conflicts though are implicitly resolved in our approach based on semantic mappings between edges. Additionally, [3] does not attempt to reduce structural redundancies and simplify the merge schema as we do in our approach.

References

1. Batini, C., Lenzerini, M., Navathe, S.: A comparative analysis of methodologies for database schema integration. ACM Computing Survers 18(4), 323–364 (1986)
2. Bernstein, P.: Applying model management to classical meta data problems. In: Proc. CIDR 2003 (2003)
3. Bernstein, P., Pottinger, R.: Merging models based on given correspondences. In: Proc. 29th VLDB (2003)
4. Boyd, M., Mçbrien, P.: Comparing and transforming between data models via an intermediate hypergraph data model. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 69–109. Springer, Heidelberg (2005)
5. Buneman, P., Davidson, S., Kosky, A.: Theoretical aspects of schema merging. In: Pirotte, A., Delobel, C., Gottlob, G. (eds.) EDBT 1992. LNCS, vol. 580, pp. 152–167. Springer, Heidelberg (1992)
6. Buneman, P., et al.: Comprehension syntax. SIGMOD Record 23(1), 87–96 (1994)

7. Jasper, E., Tong, N., McBrien, P., Poulovassilis, A.: View generation and optimisation in the AutoMed data integration framework. In: Proc. Baltic DB&IS 2004. Scientific Papers, vol. 672, pp. 13–30. Univ. Latvia (2004)
8. McBrien, P., Poulovassilis, A.: A uniform approach to inter-model transformations. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 333–348. Springer, Heidelberg (1999)
9. Poulovassilis, A., McBrien, P.: A general formal framework for schema transformation. Data and Knowledge Engineering 28(1), 47–71 (1998)
10. Rizopoulos, N., McBrien, P.: A general approach to the generation of conceptual model transformations. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 326–341. Springer, Heidelberg (2005)