AutoMed Model Management

Andrew Smith, Nikos Rizopoulos and Peter McBrien

Dept. of Computing, Imperial College London,
Exhibition Road, London SW7 2AZ

Abstract. **Model Management** (**MM**) is a way of raising the level of abstraction in metadata intensive application areas. The key idea behind Model Management is to develop a set of *generic algorithmic operators* that can be applied to a wide range of database management problems. Solutions to problems can then be specified at a high level of abstraction, by combining these operators into a concise script. The operators work on schemas and mappings between schemas, rather than individual schema elements. In this demonstration we present a new approach to the implementation of Model Management operators based on schema transformation that provides some important advantages over existing methods.

1 Introduction

Initial work in MM focused on simply on creating structural mappings between schemas, but more recent work has extended this by defining instance based semantics for the operators [1]. A system that implements the MM operators and supports the running of MM scripts is called a **Model Management System** (**MMS**). The tool presented in this demonstration, extends the AUTOMED data integration system [2] to create a transformation-based MMS that supports instance based semantics, and is capable of manipulating schemas from a wide range of data modelling languages. This confirms Berstein's conjecture [1] that the AUTOMED system is a suitable MMS candidate, and has the following advantages over existing systems:

1. ModelGen is designed in a manner that is readily applicable to a wide range of data modelling languages [3]. ER, SQL, XML and CSV schemas can be translated by the prototype used in this demonstration.
2. The implementation of Match allows for a wider set of semantic correspondences than other methods [4].
3. Mappings in our mapping language can be composed by simply adding them together, whereas in other approaches this composition is a complex task.
4. We can easily distinguish between complete and partially defined schema objects.
5. The MM operators are relatively easy to implement using our mapping language.

Points 1 and 2 have been discussed in our previous work [3, 4] and we expand briefly on points 3 to 5 in Sections 2 and 3. We describe our demonstration in Section 4.

2 BAV

We use the data integration technique **Both-As-View** (**BAV**) [5] as the mapping language in our MMS. A BAV mapping is made up of a sequence of *bidirectional* transformations that together describe precisely how instances of each **schema object** in the source schema are mapped to instances in the target schema and vice versa. Each transformation either adds, deletes or renames a single object (such as a single SQL column, SQL primary key definition, XML element, *etc*), thereby incrementally generating a new schema from an old schema. The extent of the schema object being added or deleted is defined as a query on the extents of the existing schema objects. This sequence of transformations is called a **pathway**. Example 1 illustrates a pathway with 9 transformation steps. It is important to note that the pathway describes how schema objects in *any* of the schemas in the pathway are related to the source and target schemas.

BAV has a number of advantages [5] over other mapping languages currently used in MMSs [1]. Firstly, composing mappings, the most commonly performed operation in a MMS [6], is easy. Given two BAV mappings, $s_1\_s_2$ and $s_2\_s_3$, their composition, $s_1\_s_3$ is simply the transformations in the two pathways listed together. Secondly, because BAV mappings are bidirectional, their inverse is directly available, and hence $s3\_s1$ is easily derived from $s1\_s3$. Finally, BAV allows us to differentiate between partially and completely defined schema objects, by providing two primitives for both addition and deletion. add and delete are used when we can completely define the extent of the object we wish to add or remove, and extend and contract are used when we cannot.

3 Operator Implementation

The detailed information contained in each BAV mapping enables us to implement most of the MM operators simply and efficiently. As an example, consider the Extract operator used to return the portion of a view that is derivable from a given schema. To show how using BAV as our mapping language make this operator easy to implement, consider the mapping in Example 1, used to create a view of the table employee (eid, name, dept, DoB) in our demonstration example.

Example 1. View creation pathway

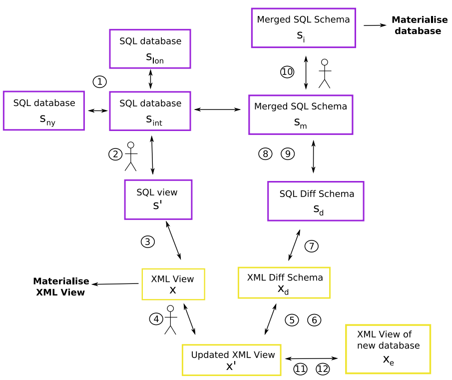
1. addTable($\langle\langle$employee_view$\rangle\rangle$, $[\{x\} \mid \{x, y\} \leftarrow \langle\langle$employee, dept$\rangle\rangle; y =' IT']$)
2. addColumn($\langle\langle$employee_view, eid$\rangle\rangle$, $[\{x, x\} \mid \{x\} \leftarrow \langle\langle$employee_view$\rangle\rangle]$)
3. addColumn($\langle\langle$employee_view, name$\rangle\rangle$, $[\{x, y\} \mid \{x\} \leftarrow \langle\langle$employee_view$\rangle\rangle; \{x, y\} \leftarrow \langle\langle$employee, name$\rangle\rangle]$)
4. extendColumn($\langle\langle$employee_view, age$\rangle\rangle$, Range $Void\ Any$)
5. contractColumn($\langle\langle$employee, DoB$\rangle\rangle$, Range $Void\ Any$)
6. contractColumn($\langle\langle$employee, dept$\rangle\rangle$, Range $[\{x, y\} \mid \{x\} \leftarrow \langle\langle$employee_view$\rangle\rangle; y =' IT']\ Any$)
7. contractColumn($\langle\langle$employee, name$\rangle\rangle$, Range $[\{x, y\} \mid \{x, y\} \leftarrow \langle\langle$employee_view, name$\rangle\rangle\ Any]$)
8. contractColumn($\langle\langle$employee, eid$\rangle\rangle$, Range $[\{x, y\} \mid \{x, y\} \leftarrow \langle\langle$employee_view, eid$\rangle\rangle\ Any]$)
9. contractTable($\langle\langle$employee$\rangle\rangle$, Range $[\{x\} \mid \{x\} \leftarrow \langle\langle$employee_view$\rangle\rangle\ Any]$)

Transformations 1 to 3 use the add primitive to create a new table and columns whose extents are the ids and names of the people in the IT department. These new schema objects are completely defined. In Transformation 4, $\langle\langle$employee, age$\rangle\rangle$ is created using the extend primitive with an extent query of Range $Void\ Any$, because we cannot determine any part of its extent from the existing schema objects. In transformations 5 to 9 the original columns and table are removed to create the view employee_view (eid, name, age) that includes only those employees from the IT department.

We can create the extract schema by extending the view creation mapping in Example 1. We know from the definition of extend that any object created with a $Void\ Any$

range cannot be derived from the source schema. The Extract algorithm therefore needs only to examine the mapping used to create the view, and perform a contract transformation on any schema objects in the view that were added using extend with a Range $Void\ Any$ query. Hence the extracted schema is employee_view(eid, name), excluding $\langle\langle$employee_view, age$\rangle\rangle$.

4 Demonstration



1. $\langle s_{int}, s_{int}\_s_{ny}, s_{int}\_s_{lon}\rangle =$ $\mathsf{Merge}(s_{ny}, s_{lon}, \mathsf{Match}(s_{lon}, s_{ny}))$
2. s_{int}_s' = User defined view creation
3. $\langle x, s'\_x\rangle = \mathsf{ModelGen}(s', XML)$
4. x_x' = User defined XML additions
5. $x'\_s_{int} = \mathsf{Invert}(s_{int}\_s' \circ s'\_x \circ x\_x')$
6. $\langle x_d, x'\_x_d\rangle = \mathsf{Diff}(x_d, x'\_s_{int})$
7. $\langle s_d, x_d\_s_d\rangle = \mathsf{ModelGen}(x_d, SQL)$
8. $s_d\_s_{int} = \mathsf{Invert}(x'\_x_d \circ x_d\_s_d) \circ x'\_s_{int}$
9. $\langle s_m, s_m\_s_d, s_m\_s_{int}\rangle = \mathsf{Merge}(s_d, s_{int}, s_d\_s_{int})$
10. $s_m\_s_i$ = User defined SQL update
11. $x'\_s_i = ((x'\_s_{int} \circ \mathsf{Invert}(s_m\_s_{int})) \oplus$ $(x'\_x_d \circ x_d\_s_d \circ \mathsf{Invert}(s_m\_s_d))) \circ s_m\_s_i$
12. $\langle x_e, x'\_x_e\rangle = \mathsf{Extract}(x', x'\_s_i)$

Fig. 1. Demonstration Example

Fig. 1 illustrates the example scenario we will show in the demonstration along with the MM script that we will run. All the operators used in the current literature [1, 6] are demonstrated here. Those that manipulate schemas: Extract, Diff, Merge, Match and ModelGen, and those used to manipulate mappings: Invert, Compose ($\circ$) and Confluence ($\oplus$). Our system includes a GUI that allows inspection and querying of all the schemas and mappings created by the script. The script can also be changed easily if necessary.

The scenario involves an organisation with offices in New York and London. They use relational databases to store the details of their employees. The New York office has a database with schema s_{ny}, while the London office has a database with schema s_{lon}.

Step 1 (Match and Merge): For administration purposes and in order to have available the details of all employees working in the organisation the local databases in NY and London have to be integrated. The Match operator returns a list of semantic mappings, which describe semantic correspondences between the objects of s_{ny} and s_{lon}, *eg*equivalence, disjointness and incompatibility [4]. These semantic mappings can be verified and improved by the person performing the integration. The Merge operator takes these semantic mappings as input and produces the integrated schema s_{int} together with data mappings between s_{ny},s_{int} and s_{lon},s_{int} [7]. The integrated schema

s_{int} can be queried to supply the details of all the employees in the organisation and other related information.

Step 2 (User Intervention): A view s' of s_{int} is created that contains only the IDs and names of the employees.

Step 3 (ModelGen): The view s' is translated into XML format using the ModelGen operator to produce x. This schema can now be sent to other departments with the organisation.

Step 4 (User Intervention): The HR department amends the schema x with the marital status and the date of birth of each employee producing schema x'.

Steps 5 and 6 (Diff): The changes to schema x are propagated to the integrated schema s_{int}. Using a combination of the Invert and Compose ($\circ$) operators the mapping from x' to s_{int} is identified. Diff takes this mapping and produces a XML schema x_d that contains the differences between x' and s_{int}.

Step 7 (ModelGen): The schema x_d is first translated into a relational schema s_d using ModelGen.

Steps 8 and 9 (Merge): The mapping between s_d and s_{int} is created using Invert and Compose ($\circ$). This is the input to the Merge operator, which produces a merged schema s_m. s_m contains the changes the HR department applied to schema x.

Steps 10 (User Intervention): The merged schema s_m is further updated to produce schema s_i.

Steps 11, 12 (Extract): In step 11, the mapping from s_i to x' is created using the $\circ$, $\oplus$ and Invert operators. This is used by the Extract operator to check that all the objects in the XML schema x' are derivable from the new database, s_i.

Apart from the fact that this demonstration is a complete implementation of data level MM operators, the fact that it is based on BAV means we are also able to demonstrate the querying of any schema in Fig. 1, based on data held in any of the other schemas in Fig. 1.

References

1. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: SIGMOD Conference. (2007) 1–12
2. M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien and N. Rizopoulos: AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. In: CAiSE04. Volume 3084 of LNCS., Springer Verlag (2004) 82–97
3. Smith, A., McBrien, P.: A generic data level implementation of modelgen. In: BNCOD. (2008) To appear
4. Magnani, M., Rizopoulos, N., McBrien, P., Montesi, D.: Schema integration based on uncertain semantic mappings. In: ER'05. LNCS, Springer (2005) 31–46
5. McBrien, P., Poulovassilis, A.: Data integration by bi-directional schema transformation rules. In: ICDE. (2003) 227–238
6. Melnik, S., Bernstein, P.A., Halevy, A.Y., Rahm, E.: Supporting executable mappings in model management. In: SIGMOD Conference. (2005) 167–178
7. Rizopoulos, N., McBrien, P.: A general approach to the generation of conceptual model transformations. In: CAiSE'05. Volume 3520 of LNCS., Springer (2005) 326–341