AutoModelGen: A Generic Data Level Implementation of ModelGen

Andrew Smith and Peter McBrien

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

Abstract. The model management operator ModelGen translates a schema expressed in one modelling language into an equivalent schema expressed in another modelling language, and in addition produces a mapping between those two schemas. **AutoModelGen** is a generic *data level* implementation of ModelGen that meets these desiderata. Our approach is distinctive in that (i) it takes a generic approach that can be applied to any modelling language, and (ii) it does not rely on knowing the modelling language in which the source schema is expressed in.

Key words: ModelGen, Model Managment, Data Transformation, Data Integration, Meta Modelling

1 Introduction

ModelGen is a model management [1] operator that translates a schema in a source **data modelling language** (**DML**), for example XML Schema, into an equivalent schema in a target DML, for example SQL, and also generates a mapping between the two schemas. To date, no implementation of ModelGen generates both a target schema and a mapping between the source and target schemas [2]. In this demonstration we present an implementation of ModelGen that automatically creates a data level mapping that describe how *instances* of the source schema should be translated [3]. Further distinguishing features of our approach are that (1) the translations are made on a **Universal Meta Model** (**UMM**) that has previously been shown to able to represent schemas from a large number of data modelling languages, and (2) the mappings created are bidirectional *i.e.* we also create a mapping from the target to the source schema.

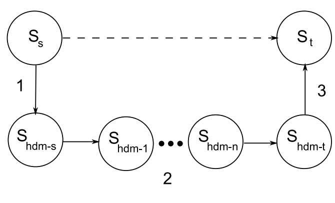

Fig. 1. Overview of the approach taken

Fig. 1 gives an overview of our approach. In step (1) the source schema S_s is transformed into an equivalent schema, S_{hdm-s} expressed in the UMM. In step (2), a series of information preserving [4] transformations are applied to S_{hdm-s} to transform it into S_{hdm-t}, that matches the structure of a schema in the target DML. In step (3) the constructs in S_{hdm-t} are transformed into their equivalents in the target DML to create S_t. We will first discuss the overall architecture of our system and then discuss the details of the two algorithms used in step (2).

2 Architecture

AutoModelGen is a tool that creates schemas and **both-as-view** (**BAV**) transformations [5] in the AutoMed data integration system [6]. AutoMed allows for schemas to be stored in both the native modelling language of a data source (*eg* XML or SQL/relational) and in AutoMed's UMM called the **hypergraph data model** (**HDM**) [7].

The HDM uses three modellings constructs (nodes, edges and constraints) to represent the constructs of a high level DML [8]. HDM nodes and edges have associated data values (called their **extent**). Constraints place restrictions on the data values that may appear in the extent. Each variant of a high level DML construct has a particular representation in the HDM. For example, the set of HDM constraints generated by a nullable SQL column will be different those generated by a not null column. This is important when it comes to identifying whether a group of HDM constructs matches a particular construct in the target DML.

A BAV information preserving [4] mapping is made up of a sequence of transformations called a **pathway**, where each transformation either adds, deletes or renames a single **schema 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.

BAV transformations can be grouped into information preserving **composite transformations** (**CT**), that act as templates of a fragment of a pathway, describing common patterns of transformation steps. For example the CT id_node_expand is useful when the target DML has explicit keys (such as a key attribute in ER or SQL models) but the source model has implicit keys (such as in XML Schema).

3 Algorithms

AutoMatch inspects a given HDM schema S_{hdm-x} and determines which of the nodes, edges and constraints match a construct in the target DML.
AutoTransform searches for a schema in which all the HDM schema objects match the structure of the target DML by repeatedly applying CTs to the schema objects in S_{hdm-x} that **AutoMatch** identifies as not matching constructs in the target DML. The set of possible schemas created in this way is called the **world**

space [9] of the problem. It can be represented as a graph whose nodes are individual HDM schemas and whose edges are the CTs needed to get from one node in the graph to the next. To limit the number of possible CTs that have to be performed at each node of the world space graph, CTs must satisfy certain preconditions before they can be executed. In our algorithm the preconditions rely on the structure of the HDM schema, in particular the constraints, surrounding the schema object that the CT is to be applied to.

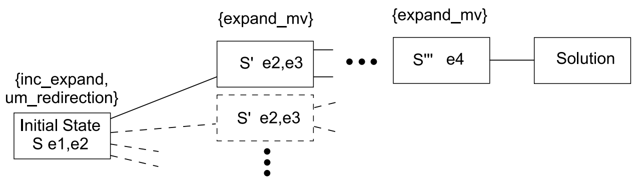

Fig. 2. An example world space graph

The world space graph for the example in the demonstration is shown in Fig. 2. Each node is labelled with a schema name $(\mathsf{S}, \mathsf{S}', \ldots$ and a list of the unidentified schema objects e1,e2,... in that schema, or the word Solution. All the constructs in a Solution node match those of the target model. Above each node is a list of CTs that meet the preconditions for the unidentified schema objects in that schema. Those CTs that meet the preconditions most closely are sorted to the top of the list and executed first. **AutoTransform** performs a depth first search on the world space graph starting from the initial state, by executing the CT at the head of the list, until a solution or a dead end is reached. If a dead end is reached the algorithm back tracks to the last node in the world space graph where an untried edge exists and executes the next CT in the list for that node. If all the edges on all the nodes have been tried without finding Solution then **AutoTransform** has failed.

4 Execution of the Tool

The current prototype of the tool is capable of translating between schemas represented in the XML, ER and SQL DMLs, and of materialising the data instances of a schema in one modelling language as instances of a second modelling language. A typical execution of the tool comprises of a user having the task of translating a schema in a source DML into an equivalent schema in some target DML, and the user performing the following steps:

1. A Source schema is imported into AutoMed, and then translated in to the HDM.
2. The **AutoMatch** and **AutoTransform** algorithms are run on the newly generated HDM schema, *hdm*, to generate a new schema *hdm'*, where the *hdm'* schema is one that matches the structure of the target DML.

3. The *hdm′* schema is translated into a schema in target DML.
4. This target schema along with its data is then materialised.

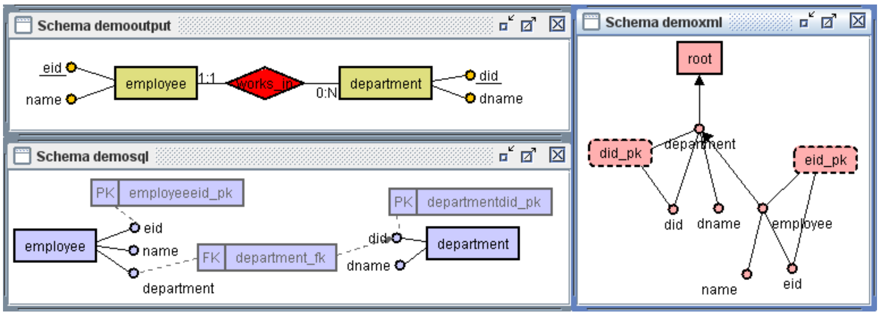

Fig. 3. Equivalent ER, SQL and XML Schemas created by **AutoModelGen**

Since the result of the tool's output is a set of schemas and BAV mappings held in AutoMed, the standard AutoMed toolkit may be used to view results of the tool's execution. Fig. 3 shows a screen shot from the AutoMed GUI featuring an ER schema and then in an anti-clockwise direction, the SQL and XML equivalents of the schema generated automatically by **AutoModelGen**.

References

1. Bernstein, P.A., Halevy, A.Y., Pottinger, R.: A vision of management of complex models. SIGMOD Record **29**(4) (2000) 55–63
2. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: SIGMOD Conference. (2007) 1–12
3. Smith, A., McBrien, P.: A generic data level implementation of modelgen. In: BNCOD. (2008) To appear
4. Hull, R.: Relative information capacity of simple relational database schemata. SIAM J. Comput. **15**(3) (1986) 856–886
5. McBrien, P., Poulovassilis, A.: Data integration by bi-directional schema transformation rules. In: ICDE. (2003) 227–238
6. 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
7. McBrien, P., Poulovassilis, A.: A general formal framework for schema transformation. In: Data and Knowledge Engineering. Volume 28. (1998) 47–71
8. Boyd, M., McBrien, P.: Comparing and transforming between data models via an intermediate hypergraph data model. J. Data Semantics IV (2005) 69–109
9. Weld, D.S.: An introduction to least commitment planning. AI Magazine **15**(4) (1994) 27–61