

Gangam: A Framework for Modeling Heterogeneous Database Transformations*

Kajal T. Claypool

University of Massachusetts-Lowell
Lowell, MA

Email: kajal@cs.uml.edu

Elke A. Rundensteiner

Worcester Polytechnic Institute
Worcester, MA

Email: rundenst@cs.wpi.edu

Key words: Cross Model Mapping Algebra, Heterogeneous System Integration, Schema Transformation

Abstract: A broad spectrum of data is available on-line in distinct heterogeneous sources, and stored under different formats. As the number of systems that utilize the heterogeneous data sources grows, the importance of data translation and conversion mechanisms increases greatly. The goal of our work is to design a framework that simplifies the task of translation specification and execution. Translation specification between the source and the target schema can be accomplished via (1) the discovery of matches between the source and the target schemata; (2) the application of a pre-defined translation templates; or (3) via manual user specification. In this paper we present a *flexible, extensible* and *re-usable* translation modeling framework wherein users can (1) explicitly model the translations between schemas; (2) compose translations from an existing library of modeled translation patterns; (3) choose from a library of translation operators; (4) generate translation models based on a match process; (5) edit such translation models; and (5) for all of these translation models, choose automated execution strategies that transform the source schema and data to the desired target schema and data. In this paper, we present the system architecture for such a translation modeling framework.

1 Introduction

Data integration and translation continue to be the impeding factors for many organizations world-wide. Today there is a broad spectrum of information that is available in interconnected digital environments such as the Web, each with its own concepts, semantics, data formats, and access methods. Currently, the burden largely falls on the human to manually (via programs) convert between the data formats, resolve conflicts, integrate the data, and interpret the results. More often than not, this barrier proves too difficult or too time-consuming to overcome and data then remains under-exploited.

The naive way to translate data from one format to another is to write a specific program for each translation task. For example, (Zhang et al., 2001; Shanmugasundaram et al., 1999) provide a specific algorithm for the hard-coded translation of XML documents into relational tuples, while other tools such as `latex2html` convert latex documents into HTML documents. In the recent past, some meta-modeling approaches have been proposed that rely on converting the input data into a common data model, and then providing a common translation language which

enables the specification and the customization of the translation logic. This approach makes the addition of new translations easier but very often still requires considerable programming effort whenever a new translation is to be defined (Bernstein and Rahm, 2000; Göbel and Lutze, 1998).

The goal of this work is to design a framework that simplifies the task of translation specification and execution. The basic observation is that, frequently, much of the structure of the source data is very similar to that of the target translated data, and many of the structure modifications to be performed by the translation process are rather standard and result from various differences between the schematic modeling capabilities of the source and the target. Moreover, there are often common patterns in the translation of source schemas of one data model to the target schemas of another data model.

Such translations between the source schema and the target schema can be generated in several different ways: (**Scenario 1**) the system can discover the matches between the source and the target schemas and present them to the user; (**Scenario 2**) the system can have a set of pre-defined translation patterns (or templates) that can be instantiated by the user for a given pair of source and target schemas; or (**Scenario 3**) the user can define the translations manually. In the case of (**Scenario 1**), the system may not

*This work was supported in part by the NSF NYI grant #IRI 97-96264.

always be able to generate matches between components of the source and target schemas; or in some cases it may discover more than one match for the same schema component. In either case, the system needs to present, using a uniform model, the matches it generates, and allow the user to manually edit and resolve the conflicts in the matches. For (**Scenario 2**), the system must provide a model to allow the user to choose from a set of given translation patterns and instantiate them for a specified source schema; while for (**Scenario 3**), it should allow users to describe their own translation between the source and the target schemas. In all three cases, it is critical to provide the building blocks using which the system can model the translation of the source schema to the target schema, irrespective of the method by which the translation was created.

Based on the above observations and requirements, we present in this paper a *flexible, extensible and re-usable* translation modeling framework wherein users can (1) explicitly model the translations between schemas¹; (2) compose translations from an existing library of modeled translation patterns; (3) choose from a library of translation operators; (4) generate translation models based on a match process; (5) edit such translation models; and (6) for all of these translation models, choose automated execution strategies that transform the source schema and data to the desired target schema and data. Note that the purpose of such a translation framework is not to *replace* the programming languages for data translations (Davidson and Kosky, 1997; Shu et al., 1975), but rather to complement them. The thrust of our work is that instead of manually writing translation programs, the system can specify them automatically using the match process or pre-defined translation patterns, or the user can model them graphically. The system can then automatically generate code, the translation programs, to transform the underlying data. Hence, the programming effort will be greatly reduced.

In this paper, we describe the translation framework. For this we describe the schema model, the building blocks that are needed to model a translation, and the architectural layers that are necessary to accomplish scenarios 2 and 3. The *match* process of scenario 1 is beyond the scope of discussion in this paper.

2 Building Blocks

The key factors that influence the achievement of a flexible and extensible translation framework are the

¹We use the term “schema” loosely to imply the structure of documents or other data sources.

building blocks that enable users to model different translations in order to transform a schema. In this section, we first present a brief overview of the common data model for the framework; and then go on to present the two main building blocks needed to model the translations.

2.1 Schema Model

```
<!ELEMENT item (location,
  mailbox, name)>
<!ATTLIST item id ID #REQUIRED
  featured CDATA #IMPLIED>
<!ELEMENT location (#PCDATA)>
<!ELEMENT mailbox (mail*)>
<!ATTLIST mailbox id CDATA>
<!ELEMENT mail (from, to, date)>
<!ATTLIST mail text CDATA>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT name (firstName,
  lastName)>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName (#PCDATA)>
```

Figure 1: A Fragment of the XMark Benchmark DTD.

We assume, as in previous modeling approaches (Atzeni and Torlone, 1996; Göbel and Lutze, 1998; Papazoglou and Russell, 1995; Bernstein and Rahm, 2000; Mark and Roussopoulos, 1983), that schemas from different data models are first represented in one common data model. In our work we assume that all schemas are represented by a simple graph called a *Sangam graph*, an instance of the Sangam graph model (Claypool, 2002). A Sangam graph $G = (N, E, \lambda)$ is a directed graph of nodes N and edges E , and a set of labels λ . Each node $n \in N$ is either complex or atomic. A complex node $n \in N$ represents a *user-defined type*, while each atomic node represents a *literal type* such as a *String* or an *integer*. Each node n is assigned a label $l \in \lambda$. Any node with only outgoing property edges is termed a *leaf*. All nodes reachable via a direct outgoing edge from a node n are called the children of the node n . Each edge $e \in E$ is either a containment or a property edge. Each edge e is annotated with a set of constraints. This includes a *local order*, that gives a monotonically increasing, relative local ordering for all outgoing edges from a given node n ; and *quantifier* that specifies the minimum and maximum occurrences of the data instances.

Figure 2 shows the Sangam graph for the XMark benchmark schema of Figure 1. Here each element and attribute is represented by a Sangam graph node. For example, the edge e_1 between the node labeled

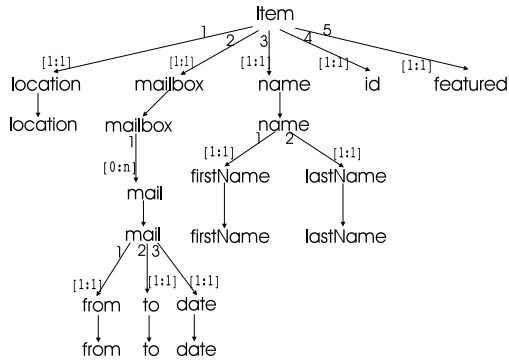


Figure 2: A Fragment of the XMark Benchmark DTD as shown in Figure 1 depicted as a Sangam Graph.

item and location represents a relationship between the item element and its sub-element location. The edge e_1 has an order annotation of 1. The quantifier annotation [1:1] on edge e_1 denotes a functional edge, i.e., that there must be exactly one object of location that can participate in a binary relationship with one object of item.

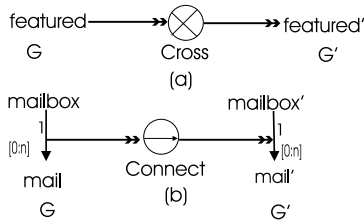


Figure 3: (a) Example of Cross Algebra Operator; (b) Example of Connect Algebra Operator.

2.2 Bricks: The Cross Algebra Operators

To enable the modeling of translations we provide two main building blocks: (1) the bricks: the cross algebra operators which allow the user to express a variety of linear transformations; and (2) the mortar: different techniques that allow users to compose the operators together to represent larger translation units. In this section we present the first building blocks, i.e., the cross algebra operators. These cross algebra operators represent the core operators, but are by no means a complete set of operators that can be provided. Other more complex operators can easily be added to extend the core functionality of the framework as described here.

In our work we have identified four core transformation operators. These operators, termed the *cross algebra operators*, represent the primitive set of op-

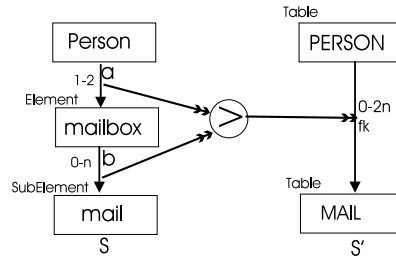


Figure 4: Example of Smooth Operator.

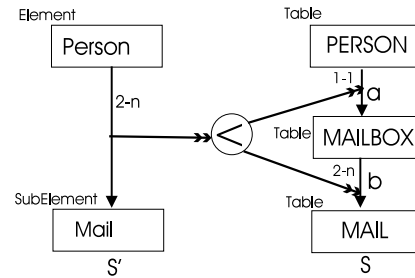


Figure 5: Example of Subdivide Operator.

erations in the class of linear graph transformations (Gross and Yellen, 1998), on the basis of which larger more complex linear transformations can be defined. The four basic algebra operators are: the *cross* operator denoted by \otimes for the addition of a node to the output Sangam graph; the *connect* operator denoted by \odot for the addition of a binary relationship between two nodes (an edge) in the output Sangam graph; the *subdivide* denoted by \oplus for splitting a binary relationship (an edge) in the input Sangam graph to a pair of binary relations (two edges) in the output Sangam graph; and the *smooth* operator denoted by \ominus , for converting two binary relations (two edges) in the input Sangam graph to one binary relation in the output Sangam graph. Translations involving the deletion of nodes or edges can simply be accomplished by not mapping the node or the edge in the input Sangam graph to the output Sangam graph. Figures 3, 4 and 5 diagrammatically depict the semantics of these operators. More details can be found in (Claypool and Rundensteiner, 2002).

2.3 The Mortar: Composition Techniques

To allow cross algebra operators to model larger, more complex transformations, in this section we now introduce two general composition techniques. Cross algebra operators can be composed into larger transformations using two techniques: (1) context dependency; and (2) derivation. We present here the main

intuition for each technique, while more details can be found in (Claypool and Rundensteiner, 2002).

2.3.1 Context Dependency Composition

The first composition technique, called the context dependency composition, enables several algebra operators to collaborate and jointly operate on subgraphs to produce one combined output graph. Figure 7 denotes such a context dependency composition CT of three cross algebra operators. Here, the algebra operators $op1_{A'}(A)$ and $op2_{B'}(B)$ are cross operators that map the nodes A and B in G to nodes A' and B' respectively in the output Sangam graph G'. The algebra operator $op3_{e'}(e)$, a \ominus operator is the root of CT and maps the edge $e: \langle A, B \rangle$ between the nodes A and B in the input Sangam graph G to the edge $e': \langle A', B' \rangle$ between the nodes A' and B' in the output Sangam graph G'. Here the outputs of all operators $op1$, $op2$, and $op3$ together produce G'.

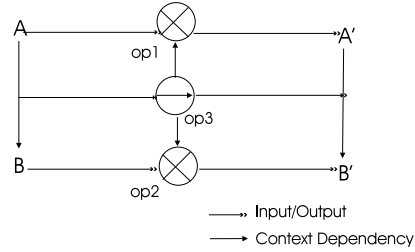


Figure 7: Context Dependency Composition Example.

3 The Architectural Layers

The schema model, cross algebra operators, and the composition techniques provide the underlying model needed to (1) represent schemas in the common schema model; (2) model the translation of a source schemata to a target schemata; (3) represent the features of different data models in a common format; and (4) model translation patterns that describe the transformation of one data model (or a fragment of a data model) to another data model.

The key thrust of our work lies in providing not only the modeling of translations between schemas, but also to provide the modeling of translation patterns which provide translations between data models. Previous research (Atzeni and Torlone, 1996; Göbel and Lutze, 1998; Papazoglou and Russell, 1995; Bernstein and Rahm, 2000; Mark and Rousopoulos, 1983; Booch, 1994) has three-layer meta-modeling presented meta-modeling architectures that provide the modeling of schemas and data models. We now extend these works and provide the modeling of not just the data models but also the modeling of translations. Figure 8 presents a three-layer approach for our transformation modeling framework. Starting at the bottom, we have the *Data* layer. This layer corresponds to the application schema and data with which we are most familiar.

The middle layer is the *Schema* layer. The application schemas of the Data layer are represented in the Schema layer using the common schema model described in Section 2. Translations are modeled in this layer using the cross algebra operators and the composition techniques described in Section 2. Each translation represents the transformation of a given source schema to a target schema. These translations can be executed to transform the data in the Data layer. Moreover, other services such as *update propagation* and *reasoning* can be built on top of such an explicitly modeled transformation.

The third layer is the *Model* layer. The Model layer captures the structural constructs of different data models such as the relational, XML and object-oriented models, and presents them using the com-

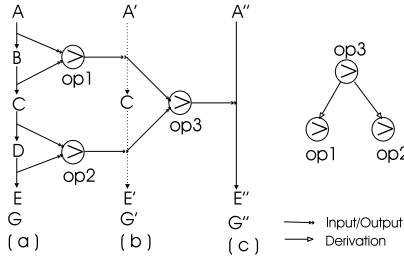


Figure 6: Derivation Composition.

2.3.2 Derivation Composition

The second composition technique is the *derivation composition*. This technique enables the nesting of several algebra operators wherein output of one or more operators becomes the input of another operator. Figure 6 gives an example of the modeling of a derivation composition that transforms the path in the Sangam graph G shown in Figure 6 (a) to the edge in the Sangam graph G' given in Figure 6 (c) by applying three smooth nodes \ominus . Let $e_1: \langle A, B \rangle$, $e_2: \langle B, C \rangle$, $e_3: \langle C, D \rangle$ and $e_4: \langle D, E \rangle$ be edges in G. Operators $op1_{e1'}(e_1, e_2)$ and $op2_{e2'}(e_3, e_4)$ are applied to the input edges e_1 and e_2 , and e_3 and e_4 respectively to first produce the intermediate edges $e_1': \langle A', C' \rangle$ and $e_2': \langle C', E' \rangle$ as shown in Figure 6 (b).

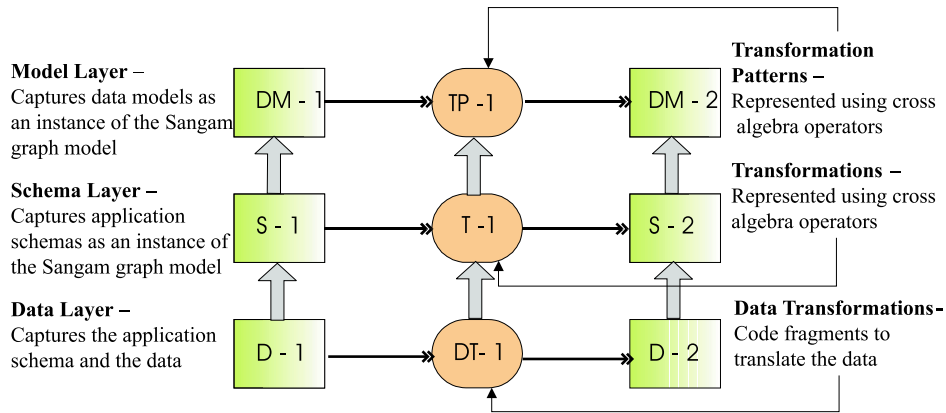


Figure 8: The Three Layered Approach for a Transformation Modeling Framework.

mon schema model (Section 2). *Translation patterns* can be presented in the Model layer using the building blocks (cross algebra operators and composition techniques) presented in Section 2. Each translation pattern represents the translation of a fragment of source data model to a fragment of a target data model. For example, an XML element may be mapped to a table in the relational model, while a sub-element may be translated to either a table or an attribute in the relational model. A set of translation patterns combined together can represent a complete translation of the source data model to the target data model. Translation patterns can be *instantiated* with a source schema from the Schema layer to produce (1) a translation in the Schema layer; and (2) a target schema based on the translation, also in the Schema layer. Thus, an instantiation of a set of translation patterns that map the XML data model to the relational data model for a given XML schema, will map the given XML schema to a relational schema as per the translation patterns. Formal description of translations and translation patterns can be found in (Claypool and Rundensteiner, 2002).

4 Gangam - The Prototype System

The Gangam system incorporating all the techniques discussed in the paper has been developed using Java technology and a variety of tools such as the JAXP (Systems, 2001) for parsing XML documents and the DTD-Parser (Wutka, 2001) for parsing the DTDs. Figure 9 gives an architectural overview of the system.

In the Model layer, we have two graphical managers, *Data Modeler* and the *Pattern Modeler*. The *Data Modeler* allows a user, most likely an administrator, to input different data models. These data

models are stored in a repository that is part of the Model layer. We are currently using Oracle8i (Corporation, 2000) to provide the storage for this and all other *data* that must be stored. *Data* here includes the translation patterns, translations, data models and the schema graphs. The *Pattern Modeler* allows the administrator to input and subsequently store the set of commonly used translation patterns. These translation patterns are modeled using the cross algebra operators and composed via the composition techniques. The *Pattern Modeler* also allows users to retrieve, compose and edit the translation patterns. A translation pattern, or a set of patterns, are instantiated via the *Instantiator* module, that takes as input the patterns to be instantiated and a source schema graph from the Schema layer. The Instantiator produces as output a translation in the Schema layer.

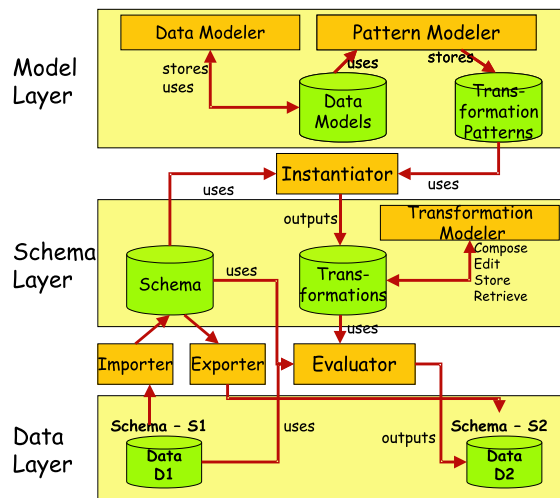


Figure 9: Architecture of Gangam

In the Schema layer, a user may import or ex-

port schemas using the *Importer* and *Exporter* modules. The functionality of these modules must be extended if and when new data models are added into the Sangam system. Users can model transformation for a given source schema via the *Transformation Modeler*. This module allows the users to retrieve existing transformations, compose new transformations, and/or edit transformations that are either retrieved from the system or instantiated by the *Instantiator* module. A transformation can be executed via the *Evaluator* module. This module takes as input the translation, the source and target schema graph and a pointer to the source data. It then transforms the data as per the translation and produces as output the transformed target data set.

Other tools sets can be added into the system at the Schema or the Model layers. Examples of such tools are update propagation tools that propagate changes from the source to the target, conformance tools that ensure that the schema graph conforms with the data model specified in the Model layer. This is especially useful for ensuring that the target schema produced by a translation conforms to a desired data model.

5 Conclusions

In this paper, we have presented our *flexible, extensible* and *re-usable* transformation modeling framework. This framework has the advantage of allowing users to model complex translations using the basic building blocks that we have presented in this paper, and then with a click of a button generating and executing code that would perform the data translation. Users of this framework can also *re-use* pre-defined translation patterns and instantiate them for their own source schemata. The translations can then be executed to perform the data transformation. A prototype of this framework has been implemented using Java, JDK 1.4, and various other Java tools. We use Oracle to handle the storage and the querying of the translation patterns, as well as the translations themselves. A version of this system was demonstrated at SIGMOD 2001 (Claypool et al., 2001).

Acknowledgments. The authors would like to thank Dr. Phil Bernstein, Microsoft Research for his valuable feedback at different stages of this project.

REFERENCES

Atzeni, P. and Torlone, R. (1996). Management of Multiple Models in an Extensible Database Design Tool. In Apers, P. M. G. and et al., editors, *Advances in*

Database Technology - EDBT'96, Avignon, France, March 25-29, LNCS. Springer.

- Bernstein, P. A. and Rahm, E. (2000). Data Warehouse Scenarios for Model Management. In *International Conference on Conceptual Modeling*.
- Booch, G. (1994). *Object-Oriented Analysis and Design*. Benjamin Cummings Pub.
- Claypool, K. (2002). *Managing Change in Databases*. PhD thesis, Worcester Polytechnic Institute.
- Claypool, K. and Rundensteiner, E. (2002). Sangam: A Transformation Modeling Framework. Technical Report UML-CS-TR-02-08, University of Massachusetts, Lowell.
- Claypool, K., Rundensteiner, E., Zhang, X., Su, H., Kuno, H., Lee, W.-C., and Mitchell, G. (2001). SANGAM: A Solution to Support Multiple Data Models, Their Mappings and Maintenance. In *Demo Session Proceedings of SIGMOD'01*.
- Corporation, O. (2000). Oracle 8i. <http://www.oracle.com/ip/dep/otn/database/8i/index.html>.
- Davidson, S. and Kosky, A. (1997). WOL: A Language for Database Transformations and Constraints. In *IEEE Int. Conf. on Data Engineering*, pages 55–65.
- Göbel, S. and Lutze, K. (1998). Development of Meta Databases for Geospatial Data in the WWW. In *ACM-GIS*, pages 94–99.
- Gross, J. and Yellen, J. (1998). *Graph Theory and its Applications*. CRC Press.
- Mark, L. and Roussopoulos, N. (1983). Integration of Data, Schema and Meta-Schema in the Context of Self-Documenting Data Models. In Davis, C. G., Jajodia, S., Ng, P. A., and Yeh, R. T., editors, *Proceedings of the 3rd Int. Conf. on Entity-Relationship Approach (ER'83)*, pages 585–602. North Holland.
- Papazoglou, M. and Russell, N. (1995). A Semantic Meta-Modeling Approach to Schema Transformation. In *CIKM '95*, pages 113–121. ACM.
- Shanmugasundaram, J., He, G., Tufte, K., Zhang, C., DeWitt, D., and Naughton, J. (1999). Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 302–314.
- Shu, N. C., Housel, B. C., and Lum, V. Y. (1975). Convert: A high level translation definition language for data conversion (abstract). In King, W. F., editor, *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 14-16, 1975*, page 111. ACM.
- Systems, J. (2001). The jaxp 1.1.1parser. <http://java.sun.com>.
- Wutka, M. (2001). The dtd parser. <http://www.wutka.com>.
- Zhang, X., Lee, W.-C., Mitchell, G., and Rundensteiner, E. A. (2001). Clock: Synchronizing Internal Relational Storage with External XML Documents. In *ICDE-RIDE 2001*.