A Model-Driven Approach to Enterprise Data Migration

Raghavendra Reddy Yeddula(✉), Prasenjit Das, and Sreedhar Reddy

Tata Consultancy Services, Pune 411 013, India
{raghavendrareddy.y,prasenjit.d,sreedhar.reddy}@tcs.com

Abstract. In a typical data migration project, analysts identify the mappings between source and target data models at a conceptual level using informal textual descriptions. An implementation team translates these mappings into programs that migrate the data. While doing so, the programmers have to understand how the conceptual models and business rules map to physical databases. We propose a modeling mechanism where we can specify conceptual models, physical models and mappings between them in a formal manner. We can also specify rules on conceptual models. From these models and mappings, we can automatically generate a program to migrate data from source to target. We can also generate a program to migrate data access queries from source to target. The overall approach results in a significant improvement in productivity and also a significant reduction in migration errors.

Keywords: Data migration · Conceptual models · Data model mapping · Query translation

1 Introduction

Data migration is the process of transferring an enterprise's data from one database to another. A typical data migration project starts with business analysts identifying the mappings between source and target data models. An implementation team then manually translates these mappings into programs that migrate the data. Business analysts specify the mappings in terms of conceptual data models that reflect the business domain semantics. Programmers have to write their programs in terms of physical database schemas. While doing this, programmers not only have to understand the source-to-target mappings correctly, but also how the conceptual models and the rules there of map to the physical databases, as shown in Fig. 1. This is an error prone process which is further compounded by the fact that these mappings are generally only documented informally, using sketchy textual descriptions.

In this paper we describe a model driven approach to automate this process using a model mappings based infrastructure. Using this infrastructure we can create conceptual models, physical models and specify mappings between them, formally, using a mappings language. This infrastructure also provides a set of primitive processing blocks such as mapping composer, query translator, etc. Mapping composer can compose a set of mappings to create a new mapping. Query translator can process a mapping to translate a query on a model at one end of the mapping to an equivalent

J. Zdravkovic et al. (Eds.): CAiSE 2015, LNCS 9097, pp. 230–243, 2015.
DOI: 10.1007/978-3-319-19069-3_15

query on the model at the other end. These blocks process not only the mappings but also the rules captured in the conceptual model.

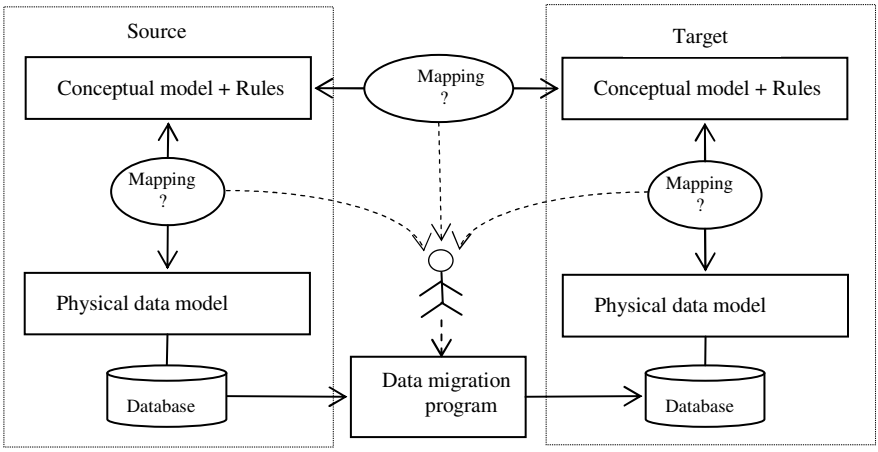

Fig. 1. Data Migration – typical scenario

In a data migration scenario, we create models and mappings as shown in Fig. 1, i.e. a mapping between source and target conceptual models, a mapping between source conceptual and physical models, and a mapping between target conceptual and physical models. Some of these models and mappings can be reverse engineered from existing systems. For instance, source-side models and mappings can be reverse engineered from an existing source system implementation, and likewise on the target side. We compose these models and mappings to automatically derive a mapping between source and target databases. From this mapping, we generate a program to migrate data from source to target. We also generate a program to migrate queries expressed on the source database into equivalent queries on the target database. The latter is important because, typically, in a data migration project, it is not only the data that has to be migrated, but also the data access programs. The overall approach results in a significant improvement in productivity. It also results in a significant reduction in migration errors.

The rest of the paper is organized as follows. Section 2 discusses the modeling infrastructure, section 3 discusses the data migration architecture and section 4 presents some results. Section 5 discusses related work. We conclude by summarizing and discussing future work.

2 Modeling Infrastructure

2.1 Conceptual Models

We use an extended version of UML object model [17] as our conceptual modeling language. Object model is extended to add rules to classes and relations. The extension to UML meta model is implemented using our in-house meta modeling tool [9].

2.2 Rules

We can specify rules on conceptual models. A rule specifies a derivative relationship or a constraint among a set of related entities. We use a rule language that is similar in flavor to Semantic Web Rule Language (SWRL) abstract syntax [18], but modified suitably to cast it in the familiar 'IF..THEN' notation that business users are more comfortable with. Also we can refer to properties and relations (i.e. object-valued properties in SWRL parlance) using path expressions which leads to a more compact and readable notation.

A rule has the following form:

```
if <antecedent> then <consequent>
```

Here, antecedent specifies a condition on a set of related entities and the consequent specifies the inferences one can draw when the condition holds. Detailed explanation of this language is outside the scope of this paper. We give a few examples to illustrate the syntax. Fig. 2 shows a simple conceptual model where `Person` is a concept and `parent`, `spouse` and `sibling` are relationships.

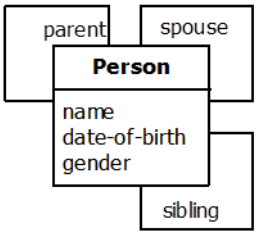

Fig. 2. A simple conceptual model

The following are some of the rules and constraints that one can express on this model:

Rule 1: Persons who have a common parent are siblings.

```
If
  Person(p1) and Person(p2) and p1.parent = p2.parent
Then
  p1.sibling = p2
```

Constraint 1: Two different parents (e.g. mother and father) of a person cannot have the same gender.

```
If
  Person(p1) and Person(p2) and Person(p3)and
  p1.parent = p2 and p1.parent = p3 and p2 <> p3
then
  p2.gender <> p3.gender
```

2.3 Physical Models

We use relational model for physical data modeling. A simplified version of the relational meta model is shown in Fig. 3. Using this model we can specify relational tables and primary-key and foreign-key relations among them.

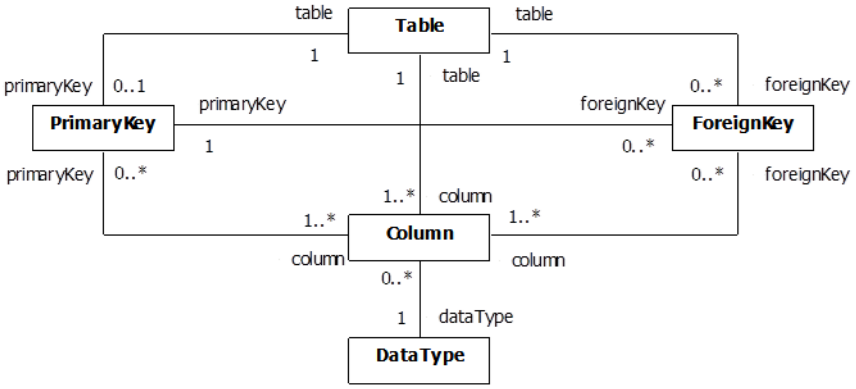

Fig. 3. Relational meta-model

2.4 Model Mappings

We map two models by defining a class or a table of one model as a view over one or more classes or tables of the other model. Views are defined declaratively using a language that is a restricted version of Object Query Language (OQL) [19]. We call this language PQL (for path expression query language). This is essentially OQL

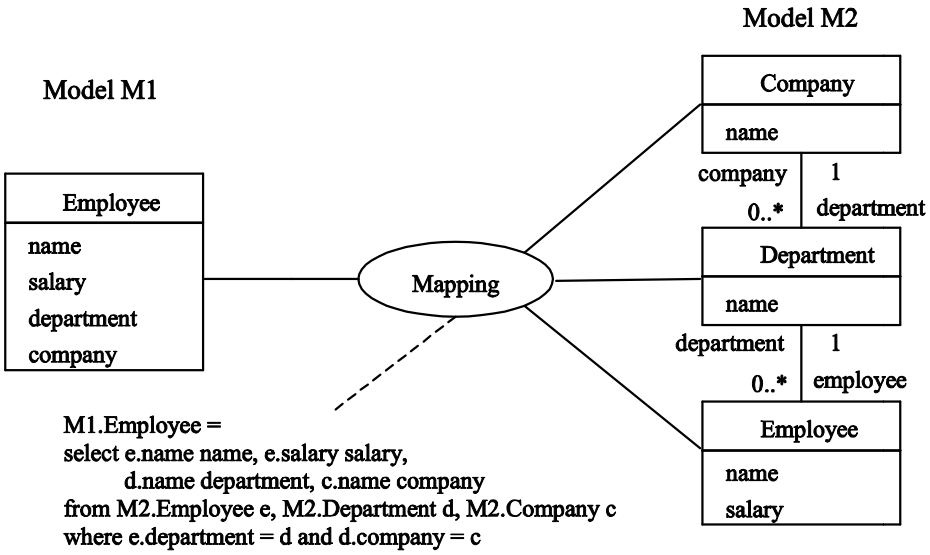

Fig. 4. Sample Mapping

without some of the procedural extensions such as method invocations on objects. We support user-defined functions but they are required to be side-effect-free, i.e. they just take inputs and return an output without leaving any side-effect in the state.

Fig. 4 shows an example mapping where a class in model M1 is mapped as a view over a set of classes in model M2.

2.5 Model Processing Infrastructure

We use an internal representation language that is similar to Datalog with aggregation [1], [3], [20]. We translate PQL queries, mappings and rules into this internal representation language. Detailed discussion of this language is outside the scope of this paper due to space constraints, but the following examples will give an idea.

Example 1
PQL (mapping of M1.Employee in Fig. 4):

```
select e.name name, e.salary salary, d.name department,
       c.name company
from M2.Employee e, M2.Department d, M2.Company c
where e.department = d and d.company = c;
```

Internal Representation

```
M1.Employee(name(v1), salary(v2), department(v3),
  company(v4)) :-
    M2.Employee(id(v0), name(v1), salary(v2)),
    M2.employee_department(id1(v0), id2(v5)),
    M2.Department(id(v5), name(v3)),
    M2.department_company(id1(v5), id2(v6)),
    M2.Company(id(v6), name(v4)).
```

Example 2
Rule (on M1.Employee)

```
If
  Employee(e) and e.salary > 120000
Then
  e.department = 'Management'
```

Internal Representation

```
Employee(id(v0), name(_), salary(v1),
  department('Management'), company(_)) :-
    Employee(id(v0), name(_), salary(v1), department(_)),
    v1 > 120000.
```

Please note that Datalog does not support functional terms in arguments. Our internal notation does not support them either. References to functional terms such as id(), salary(), etc. above are added purely for the sake of readability, to show which arguments refer to which data elements. In the actual implementation, we only use

variable arguments, where the position of an argument uniquely identifies the data element it represents.

Also note the usage of object IDs (id() terms), and the codification of a relationship as a term with two ID arguments. For example, the relationship between Employee and Department is represented by the term employee_department(id1(), id2()).

With respect to a given model M, we classify the internal representation rules of mappings into two classes, the so-called global-as-view (GAV) [2], [5] and local-as-view (LAV) rules [4]. Rules of a mapping where a class of M is defined as a view over classes of other models are classified as GAV rules; whereas rules of a mapping in the other direction, i.e. a class of the other model being defined as a view over classes of M, are treated as LAV rules.

The uniform representation of rules and mappings allows us to process them together. For instance, referring to the rule above, if we have a query asking for employees of 'management' department, we can return all employees whose salary is greater than 120000.

2.6 Query Translator

We use a query translation algorithm to translate queries written on a model at one end of a mapping into an equivalent query on the other end of the mapping, as shown in Fig. 5. The algorithm is based on the well-known GAV [2] and LAV [4] query rewriting techniques. A detailed discussion of the query translation algorithm is outside the scope of this paper. A detailed discussion on the individual techniques can be found in the cited references. We combine these techniques in a layered manner, where each layer does a partial translation.

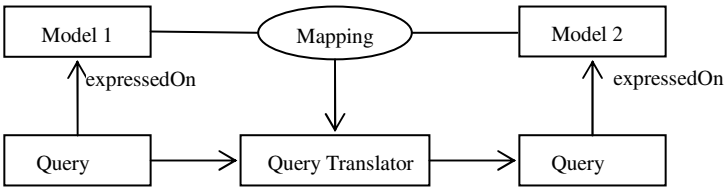

Fig. 5. Query Translation

2.7 Data Flow Graph Generator

A Data Flow Graph or DFG represents the flow of data from one system to another system and the transformations it undergoes along the way. A DFG contains nodes and directed edges. Nodes represent operators. An edge represents flow of data from one node to the other. There are various types of operators in a DFG, such as *join, union, filter, aggregation*, etc. each performing a specific operation. For example, *join* operator joins the data tuples coming in on its input edges and sends out the joined tuples on its output edge. A data flow graph is a procedural artifact. We can either

execute it directly or translate it into procedural code in a standard high-level language such as java, stored procedures, etc.

We can translate a PQL query into an equivalent data flow graph. We first translate the query into our internal Datalog representation, and then generate the DFG from this internal representation. We illustrate this with a few examples:

Example 1
Datalog Query

```
Customer(name(v1)) :- CorporateCustomer(name(v1), ..).
Customer(name(v1)) :- IndividualCustomer(name(v1), ..).
```

The query is used to fetch names of customers from two sources -- corporate customers and individual customers by executing the corresponding sub queries `CorporateCustomer()` and `IndividualCustomer()`.

Fig. 6 shows the equivalent DFG. The boxes `CorporateCustomer` and `IndividualCustomer` represent the sources. The Selection operators select name of `CorporateCustomer` and `IndividualCustomer` from respective sources.

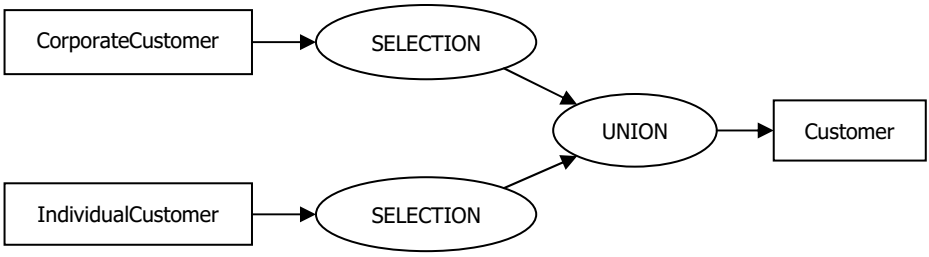

Fig. 6. DFG – Example 1

Example 2
Datalog Query

```
MillionDollarCustomer(name(v1), amount(v2)) :-
  SQ(v1, v2), v2 > 1000000.
SQ(v1, SUM(v3)) :-
  Customer(id(v0), name(v1), ..),
  Customer_Contract(customer(v0), contract(v2)),
  Contract(id(v2), amount(v3)).
```

This query is used to fetch names and total contract amount of customers whose total contract amount exceeds a million.

Fig. 7 shows the equivalent DFG.

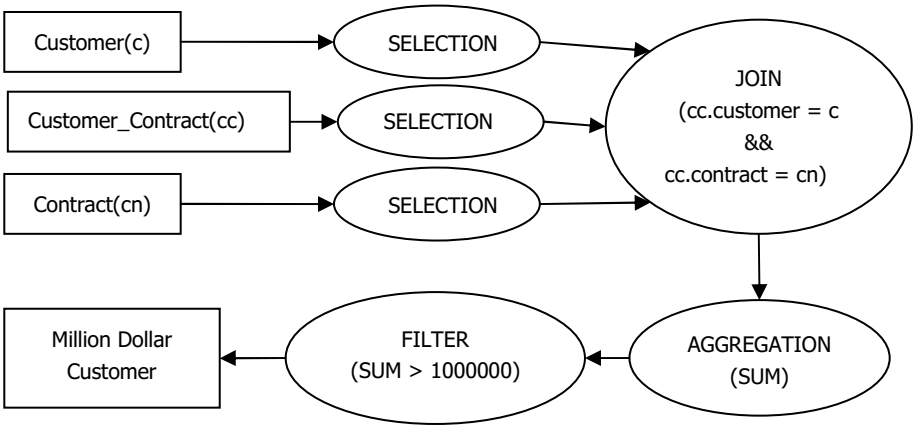

Fig. 7. DFG – Example 2

2.8 Mapping Composer

We can derive a mapping between two unmapped models by composing other known mappings. Referring to Fig. 8 below, suppose we have three models M1, M2 and M3. Suppose we know the mappings between M1 and M2, and between M2 and M3, namely MAP1 and MAP2. We can compose MAP1 and MAP2 to derive a new mapping MAP3 between M1 and M3.

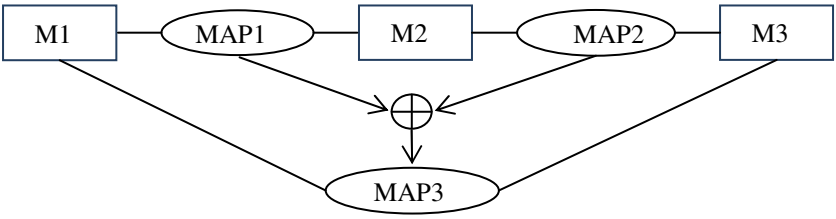

Fig. 8. Mapping composition

Composition is done by a series of query translation steps. Suppose MAP1 specifies a class C1 in M1 as a view V1 over a set of classes in M2, and MAP2 in turn specifies classes in M2 as views over classes in M3. Then a translation of the view query V1 from model M2 to model M3 gives the mapping of class C1 in terms of classes in M3.

3 Data Migration Solution

Fig. 9 below depicts the data migration solution implemented using the modeling infrastructure discussed in the previous section. We define source and target conceptual

models, corresponding physical models, and the mappings between them as depicted in the figure. We can reverse engineer physical models from database schemas. We can create initial versions of conceptual models (where they do not explicitly exist) in one of the following ways:

- Derive a default one-to-one conceptual model from the physical model using a set of canonical object-relational mapping patterns. For example, a table becomes a class, a column becomes a property, a foreign-key between tables becomes a relation between the corresponding classes, etc.
- If a system is implemented using object-oriented technology, then we can reverse engineer the object model as the conceptual model and the object-relational mapping as the conceptual-physical mapping.

These initial conceptual models are then refined in consultation with domain experts. Data migration team then defines the mapping between source and target conceptual models. We can use a schema matching algorithm [15, 16] to discover initial correspondences. These are then refined into mappings in consultation with domain experts. From these mappings, we automatically derive a mapping between source and target physical models using mapping composer. Referring to Fig. 9, we derive the mapping PP_MAPPING by composing S_CP_MAPPING, CC_MAPPING and T_CP_MAPPING.

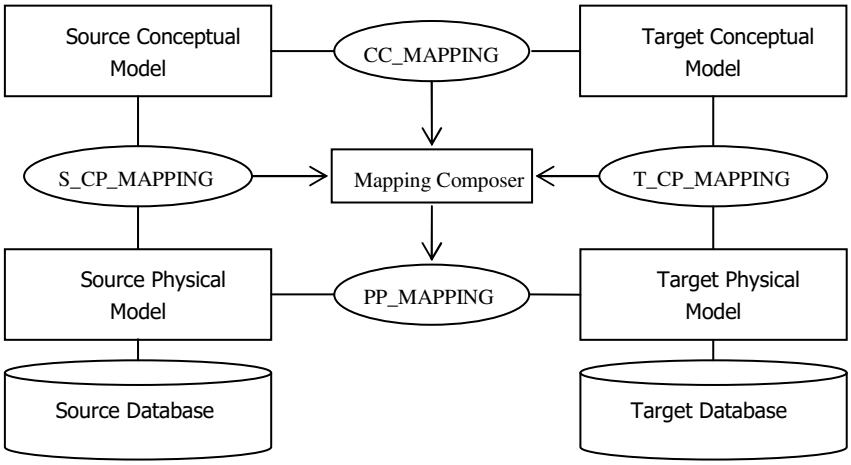

Fig. 9. Data Migration – modeling schema

A point to note here is that there is no inherent limitation on the number of source databases that can be mapped to the source conceptual model. We can have more than one source database, and so more than one source physical model and corresponding physical-conceptual mappings. This is typically the scenario when data is being migrated from a number of related databases.

3.1 Generating Data Migration Program

Conceptually the data we need in a target database table T is the data that should satisfy the query 'select * from T'. So for each target table T we start with this query. Then we translate this query into an equivalent query on the source data models. From the translated query we generate a data flow graph. This process is depicted in Fig. 10.

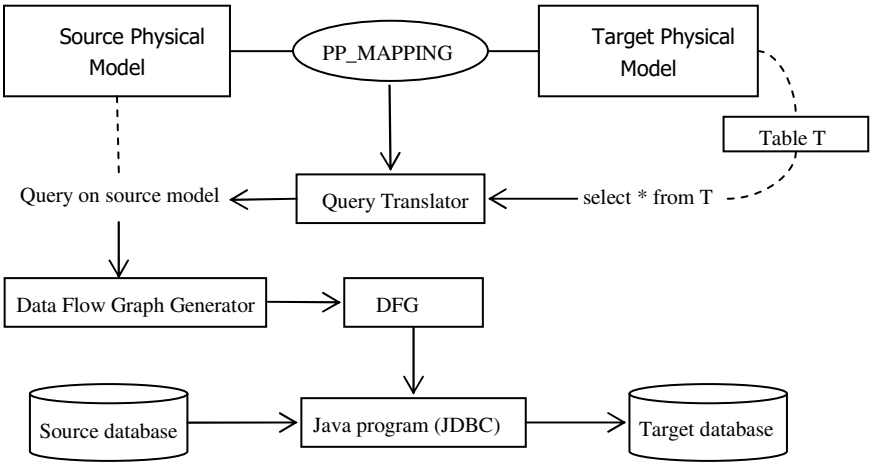

Fig. 10. Generating data migration program

We can execute the generated DFGs directly to transfer source data to the target tables, or we can translate them into platform specific ETL [8] processes. We can also translate the DFG to a platform specific executable program. For example, we can translate the DFG to a program in Java and JDBC.

We also generate a master program that invokes the individual programs in an order that honors referential integrity constraints in the target database. For instance, in a parent-child relationship, the DFG of the parent table is invoked before the DFG of the child table.

3.2 Migrating Queries

In a data migration project, typically, it is not only the data that has to be migrated, but also the data access programs. Query migration is at the heart of data access program migration. We can use the derived mapping PP_MAPPING to translate queries on the source side into equivalent queries on the target side, as shown in Fig. 11.

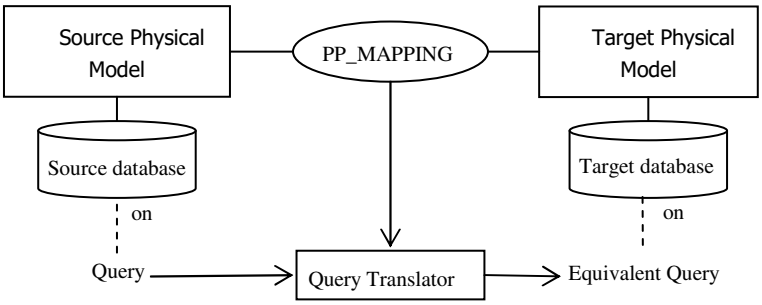

Fig. 11. Migrating Queries

For example, we take the following query on model M1 in Fig. 4.

```
SELECT company, SUM(salary)
FROM Employee GROUP BY company;
```

We generate the following equivalent query on model M2.

```
SELECT v2.name, SUM(v1.salary)
FROM Employee v1, Company v2
WHERE v1.department.company = v2 GROUP BY v2.name;
```

We generate a program that internally uses the query translator and the composed mapping to migrate source queries to the target.

4 Results

We tested our data migration approach in a product migration scenario in the financial services business domain. Our company has developed a financial services product. Deploying this product in a customer organization involves migrating a customer's existing systems to our product. Data migration is one of the first steps in the overall migration process. Customers' data is typically present in customer specific data formats and our financial services product stores its data in its own data model.

We took the case of one of our customers whose data was migrated to this financial services product. The migration was done by writing a set of custom PL/SQL programs. The customer's data was spread across two databases, with each database having more than 170 master tables. The record count in the tables ranges from a few hundred to a few millions.

Using the data migration tool, we reverse engineered the physical data models from the database schemas of customer's databases and our product database respectively. We also created corresponding conceptual models. Mappings were then identified between source and target models. Mapping specifications were of different

complexities, varying from simple one-to-one mappings to complex mappings involving join conditions and sub-queries.

We then generated the data migration program to migrate data from customer's databases to our product database. Using the data migration approach discussed earlier DFGs were generated for target tables. Data migration program was then generated from these DFGs. This program had more than 400 Java Classes. This program was executed and results were compared with the data obtained from the previous hand-coded approach.

Preliminary assessment suggests that approximately 1 person year has been spent on this tool based migration approach. The corresponding effort in the old approach of creating custom PL/SQL programs was more than 5 person years. This is around 80% improvement in productivity. Based on the analysis of available defect logs from the traditional approach, our initial estimate of error reduction is around 30%. These improvements are primarily due to automated code generation from model mappings, thereby improving productivity and eliminating error-prone manual coding. Model mappings are also much easier to verify, leading to early detection of errors.

5 Related Work

Commercial data migration tools [12, 13, 14] provide a graphical environment where developers create an ETL [8] process. These tools provide a library of operators along with an execution engine. ETL processes are essentially platform specific variants of data-flow graphs discussed in this paper. ETL based approaches suffer from the same issues discussed in this paper, viz., mappings are identified at a conceptual level, and a programmer has to understand how the conceptual models and their rules map to physical models and then translate this understanding into data-flow graphs, which is an error-prone and effort-intensive process [10].

In [6] Simitsis et al propose an approach that uses semantic web technologies for designing ETL processes. They annotate data elements of data sources and warehouse using a common vocabulary, and from these annotations infer data transformations required to populate the warehouse from data sources. While this is an interesting approach, our experience suggests that it does not scale up for complex industrial scale problems. We need full-fledged conceptual models with rules and full-fledged mappings to capture the semantics. Annotations are useful but not sufficient. They also talk about generating an ontology from such annotations. This again is a useful feature to have when there is no explicitly defined ontology. Indeed we ourselves use a similar approach when we derive a conceptual model from a physical model as explained in section 3. Again, in our experience, this can only give an initial simplistic version, which has to be refined subsequently in consultation with domain experts.

In [7] Lilia et al propose a Model Driven Architecture (MDA) based framework for development of ETL processes. They talk about specifying ETL processes at two levels -- a platform independent model (PIM) (they call it a conceptual model) and a platform specific model (PSM). They use UML activity diagrams to specify ETL processes at the PIM level and use Query/View/Transformations (QVT) specifications

to transform them into platform specific ETL models. While this gives a measure of independence from platform level variations, it does not significantly raise the level of abstraction. Activity diagrams are still procedural artifacts. They are not easy to reason with to support operations such as query translation.

In [11] Dessloch et al discuss Orchid, an approach for integrating declarative mappings with ETL processes. Their approach is the closest to our approach in spirit. They also talk about declarative mappings between models and generating ETL specifications from these mappings. However, they propose an operator model as the internal representation, where as we propose a logic based notation as the internal representation. Logic based notation allows us to treat both mappings and rules uniformly, enabling us to process them together. The abstract operator model proposed by Orchid is similar to the model we use for representing data-flow graphs.

[15, 16] provide surveys of schema matching approaches. As discussed in section 3, these approaches provide initial correspondences between models. These have to be refined into mappings in consultation with domain experts.

6 Conclusion and Future Work

A mapping document that specifies relationships between source and target data elements is the starting point for most data migration implementation efforts. We have shown that using our modeling framework we can specify these mappings formally, at a conceptual level, closer to the business domain, and use model driven techniques to automatically generate data migration programs. The automation helped us eliminate manual efforts in various stages of data migration, thereby increasing productivity and reducing the chances of errors. We have also shown how queries can be migrated. We plan to extend this approach to migrate stored procedures and embedded queries by integrating it with a program migration framework. We also plan to explore how industry standard reference conceptual models, such as ACORD for insurance [21], can be exploited to facilitate data exchange among applications. These standard models are growing in popularity and many applications are adopting them as their reference conceptual models. Lack of good conceptual models is one of the stumbling blocks in adopting model driven approaches such as the one discussed in this paper. Adopting industry reference models addresses this problem. We map application specific models to the industry reference model. From these mappings we can automatically derive mappings between any two applications, and use them to facilitate data exchange between the applications.

References

1. Abiteboul S., Hull R., Vianu V. Foundations of Databases. Addison Wesley, Reading, Mass., USA (1995)
2. Ullman, J.D.: Information integration using logical views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1996)

3. Ullman, J.D.: Principles of database and knowledge-base systems, vol. I, II. Computer Science, Rockville, Md., USA (1989)
4. Halevy, A.Y.: Answering queries using views: A survey. The VLDB Journal 10(4), 270–294 (2001)
5. Maurizio, L.: Data integration: a theoretical perspective. In: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. ACM (2002)
6. Skoutas, D., Simitsis, A.: Designing ETL processes using semantic web technologies. In: Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP. ACM (2006)
7. Muñoz, L., Mazón, J.-N., Trujillo, J.: Automatic generation of ETL processes from conceptual models. In: Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP. ACM (2009)
8. Vassiliadis, P., Simitsis, A.: Extraction, transformation, and loading. Encyclopedia of Database Systems. Springer US, pp. 1095-1101 (2009)
9. Kulkarni, V., Reddy, S.: A model-driven architectural framework for integration-capable enterprise application product lines. In: IEEE European Conference on Model Driven Architecture - Foundations and Applications, Bilbao, Spain, July 2006
10. Dayal, U., et al.: Data integration flows for business intelligence. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM (2009)
11. Dessloch, S., et al.: Orchid: integrating schema mapping and etl. In: IEEE 24th International Conference on Data Engineering, ICDE 2008. IEEE (2008)
12. Abinitio, March 2014. http://www.abinitio.com/#prod-cs
13. Informatica, March 2014. http://www.informatica.com/in/solutions/enterprise-data-integration-and-management/data-migration/
14. Talend, March 2014. http://www.talend.com/solutions/data-migration
15. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal (2001)
16. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 146–171. Springer, Heidelberg (2005)
17. Unifed Modeling Language. www.omg.org/spec/UML
18. Semantic Web Rule Language. http://www.w3.org/Submission/SWRL/#2
19. Object Query Language. Wikipedia. en.wikipedia.org/wiki/Object_Query_Language
20. Cohen, S., Nutt, W., Serebrenik, A.: Algorithms for rewriting aggregate queries using views. In: Masunaga, Y., Thalheim, B., Štuller, J., Pokorný, J. (eds.) ADBIS 2000 and DASFAA 2000. LNCS, vol. 1884, pp. 65–78. Springer, Heidelberg (2000)
21. ACORD. https://www.acord.org/