Database Wrappers Development: Towards Automatic Generation

Philippe Thiran
Technische Universiteit Eindhoven
The Netherlands
ph.thiran@tue.nl

Jean-Luc Hainaut
Université de Namur
Belgium
jlh@info.fundp.ac.be

Geert-Jan Houben
Technische Universiteit Eindhoven
The Netherlands
g.j.houben@tue.nl

Abstract

Wrapping databases allows them to be reused in formerly unplanned contexts, such as web-based applications or federated systems. Indeed, a wrapper can provide external clients of an existing (legacy) database with a neutral interface and augmented capabilities. However, except in simplistic cases where the wrapper and the database schemas are similar, the wrapper must implement complex mappings: it must translate queries from the wrapper data manipulation language to the database primitives, and, conversely, translate extracted data into the external wrapper format. We have developed a generic schema mapping framework in which wrappers can be specified formally and generated automatically. This framework comprises a high-level generic data model and a set of schema transformations defined for this model. This reference model makes it possible to specify different data models in a uniform formalism. Mappings between schemas are expressed as sequences of reversible schema transformations. We show how these transformations can be used to translate data and queries between two schemas and hence to generate as much as possible of the code of the wrappers. The generation is supported by DB-MAIN, a wide-spectrum CASE tool.

1. Introduction

Existing data systems contain vital information that is embedded in existing (most often legacy) databases/ flat files and application code. In many cases, data systems include the only source of years of business rules and other valuable information. Access to this information is of vital importance to new open environments like the Web and to system integration in general.

A wrapper attempts to extend the usefulness of components of the existing data systems by facilitating their integration into modern (distributed) systems. A wrapper addresses the challenge of database heterogeneity by providing a standard and common interface. This interface is made up of: (1) a *wrapper schema* of the wrapped database, expressed in a canonical data model and (2) a common query language which uses the semantics defined in the wrapper schema. Queries on the wrapper schema are also known as *wrapper queries*.

Basically, database wrapping involves two, generally different, models, namely the database model (e.g., relational or standard files) and the wrapper model (e.g., object-oriented or XML). The main function of a wrapper is the translation of queries posed on the wrapper schema to the database model, and, conversely, the translating of data from the database model to the wrapper model.

1.1. Proposal

This paper focuses on the aspects of query and schema translation within wrappers. We consider a generic schema mapping framework in which wrappers can be specified formally and generated automatically. This framework comprises a high-level generic data model and a set of schema transformations defined for this model. This reference model makes it possible to specify query and schema mappings in a uniform and unique formalism.

In this paper, we extend the work in [16] and [17] on wrappers for legacy databases by exploring the query and schema mappings of such wrappers. [16] introduces the concept of wrappers that enact complex database/wrapper schema mappings, and describes the general architecture and functions of such wrappers. [17] addresses the problem of consistency control when wrappers are allowed to update data. The main contribution of this paper is the detailed development of the mechanism through which schemas and queries are translated from the wrapper schema to the database schema, and conversely.

1.2. Related Approaches

Several research projects have already investigated this issue related to query mappings. Unlike their approaches, we investigate the problem from a model-independent and schema-oriented perspective:

- *Model-independent perspective.* Current approaches for wrapping databases rely on couples of models, such as those intended to produce XML views of relational schemas ([2], [3] or [13]). In this work, we use a general formalism to reversible schema transformations [7] based on a generic high-level data model. It provides a formal and uniform description of arbitrary models and the use of schema conversions between two not necessarily equivalent models. Here we extend this work by using our schema transformations to automatically wrap queries and data between two schemas.
- *Schema-oriented perspective.* Considering the issue of mapping definition and according to [12], two main basic approaches have been used to specify them. The first and very widespread approach ([1], [2], [3], [4] or [10]) is *query-oriented* in that it provides mechanisms by which users define wrapper schema constructs as views over source schema constructs, but do not focus on the semantics of the data sources. In contrast, the second approach ([11] or [14]) is *schema-oriented* in that mappings are defined as schema transformations that are used to automate the translation of queries. A comparison of these approaches is reported in [12]. The schema-oriented approach has the further advantage of decomposing the transformation of schemas into a sequence of small steps, whereas the query-oriented approach requires to directly define constructs in one schema in terms of those in the other schema.

We differ from the approach of [11] in using a high-level generic data model instead of a low-level one. [11] defines data structures as semantics-free binary graphs (made up of a very small set of nodes, edges and names). This approach is *constructive* (or bottom-up) in that operational models and transformations are built by assembling elementary building blocks. The approach we propose here is based on high-level data model that includes a greater variety of constructs, each of them being a natural abstraction of one or several constructs of lower-level models. This approach is qualified *by specialization* (or top-down) in that an operational model and its transformational operators are defined by specializing (i.e., selecting, renaming, restricting) constructs and transformations. A comparison of these approaches is given in [7]. The key advantage of the *specialization* approach is that the transformations within an operational model are those of the generic model which remain meaningful whereas the constructive approach requires to define specific transformations for each operational model.

1.3. Paper organization

The paper is organized as follows. Section 2 presents the generic mechanisms of query mappings within a transformational approach. We then present in Section 3 the high-level generic data model that underpins our approach and the primitive transformations on schemas defined in terms of this model. In Section 4, we show how the schema transformations that result from our framework can be used to automatically translate queries. Section 5 deals with the development of databases wrappers based on schema transformations and its CASE support. Section 6 presents some metrics of the wrapper development cost. They illustrate the necessity of the wrapper development/construction. Finally, Section 7 concludes this paper.

2. Transformational approach of query mapping

Query translation is the core function of a wrapper. It refers to operations that translate queries between two schemas (the database and wrapper schemas) and two languages (the database[1] and wrapper query languages).

Considering the issue of translating queries from one language to another one, our idea is to use an intermediate level, independent of all the possible operational query languages. We therefore use an internal abstract query language as the bridge for the translation rather than directly translating wrapper queries into database queries.

Considering the issue of schema mapping, our approach is to use schema transformations that provide mechanisms for formally defining the schema correspondence between the database and wrapper schemas, and then, on using that equivalence to automatically perform the query mappings.

Figure 1 shows the translation process[2]. The wrapper query `Q1` is first stripped off, creating an internal form `Q2` that captures purely the semantics of the query. Next, `Q3`

1 Or more precisely the Data Management System (DMS) language, generally called Data Manipulation Language, or DML for short.

2 For simplicity, we make the hypothesis that the wrapper query language and the internal query language are equivalent. We refer to [5] for a discussion about the problem of query equivalence.

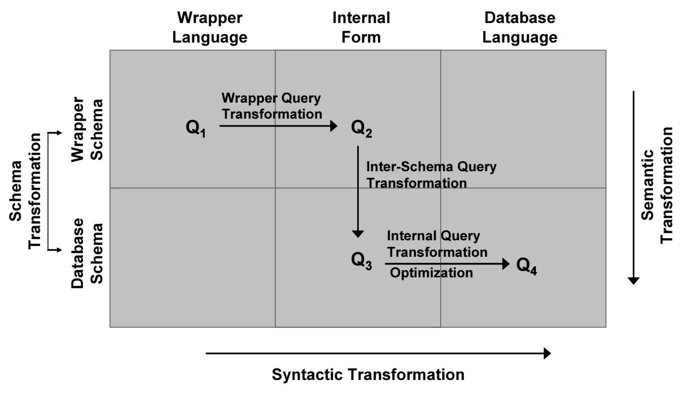


Figure 1. Language and schema mappings of a wrapper query Q1 into a DML query Q4.

is derived by application of the schema transformations on the constructs of Q2. Finally, Q3 is translated into a query Q4 that complies with both the database schema and the database DML.

We can now state the three main successive steps of query translation:

- *Language mappings:* syntactic translation of the wrapper query into an internal form.
- *Inter-schema mappings:* semantics translation of the query using the schema transformation approach for defining the mappings between the database and wrapper schemas.
- *Language mappings and optimization:* syntactic translation of an internal form into a query based on the DBMS query language. Producing an efficient execution strategy depends on the syntax and expressiveness of both the wrapper (or internal) and DBMS query processing capabilities. Dealing with such issues is out of the scope of this paper however. In [16], we present some strategies for implementing query processors and optimizers in wrappers dedicated to COBOL systems.

In Section 3, we describe the formal framework of reversible transformations based on a generic data model. In Section 4, we present the internal query language based on the same data model. As we will see through these sections, reversible transformations allow internal queries to be automatically translated in either direction between two schemas.

3. Generic transformational framework

Query translation is a process that relies on mappings between schemas that are built within different paradigms. In the proposed approach, database and wrapper schemas are expressed in a unique wide spectrum specification model, the so-called *Generic Entity-Relationship model* (GER), from which the operational data models can be derived by specialization, that is, by selecting a subset of concepts and by defining restrictive assembly rules. As a result, it provides an ideal support for our query translation approach based on schema transformations. Indeed, any transformation can be used whatever their underlying data model. For instance, the same schema transformation can be used in a relational schema and in an ER schema.

This section gives a short overview of the model and of the transformation techniques. More details of this approach can be found in [14].

3.1. Generic Entity-Relationship model

For the need of this paper, the GER can be perceived as an enriched variant of the standard entity-relationship model. It includes the concepts of *entity type*, *attribute*, *value domain* and *relationship type*. Attributes can be atomic or compound, mandatory or optional, single-valued or multivalued. The roles of a relationship type can be labelled; it has a cardinality constraint (a pair of integers stating the range of the number of relationships in which any entity can appear). An attribute has a cardinality constraint too, that states how many values can be associated with each parent instance (default is 1-1 and does not appear in graphical schemas). In general, several properties hold, and must be declared, among the components of an entity type: uniqueness, referential and existence constraints are just some of them. Due the wide variety of such properties, the GER includes the generic concept of property *group*, or group for short. A group is any subset of components (attributes and/or roles) of an entity type on which one or several properties are defined. The label(s) of the group specifies its properties (id for identifier, ref for referential, excl for exclusive, and so on). For example, a group of attributes of entity type E can be declared identifier and referential. This group models such relational pattern as a primary key that simultaneously is a foreign key.

This generic data model can be specialized into any operational model. A specialized model is built by selecting generic constructs and structural constraints, and by renaming constructs to make them comply with the concept taxonomy of the specialized model. As an illustration, the relational model, considered as an operational database model, can be precisely defined as follows (standard ER, UML class diagrams, IMS, Cobol, OO or XML DTD and Schema[3] can be defined in the same way):

- *Selecting constructs.* We select the following constructs: entity types, domains, attributes, identifiers and reference attributes.
- *Structural constraints.* An entity type has at least one attribute. The valid attribute cardinalities are `[0-1]` and `[1-1]`. An attribute must be atomic.
- *Renaming constructs.* An entity type is called a table, an attribute is called a column, an identifier, a key and a group of reference attributes, a foreign key.

3.2. Mapping definition

A transformation consists in deriving a target schema `S'` from a source schema `S` by replacing construct `C` (possibly empty) in `S` with a new construct `C'` (possibly empty).

More formally, considering instance `c` of `C` and instance `c'` of `C'`, a transformation Σ can be completely defined by a pair of mappings `<T,t>` such that `C' = T(C)` and `c' = t(c)`. `T` is the structural mapping, that explains how to replace construct `C` with construct `C'` while `t`, the instance mapping, states how to compute instance `c'` of `C'` from any instance `c` of `C`.

3.2.1. Inverse transformation. Each transformation $\Sigma_1 \equiv$ <T_1,t_1> can be given an inverse transformation $\Sigma_2 \equiv$ <T_2,t_2>, usually denoted Σ^{-1}, such that, for any structure `C`, $T_2(T_1(C)) = C$.

So far, Σ_2 being the inverse of Σ_1 does not imply that Σ_1 is the inverse of Σ_2. Moreover, Σ_2 is not necessarily reversible. These properties can be guaranteed only for a special variety of transformations[4], called symmetrically reversible. Σ_1 is said to be a symmetrically reversible transformation, or more simply semantics-preserving, if it is reversible and if its inverse is reversible too.

From now on, unless mentioned otherwise, we will work on the structural part of transformations, so that we will denote a transformation through its `T` part.

3.2.2. Some typical transformations. We propose in Figure 2 the most common transformational operators. In particular, these transformations are sufficient to carry out the transformation of most ER schemas into the relational schemas [6], and conversely. Experience suggests that a collection of about thirty of such techniques can cope with most database engineering processes, at all abstraction levels and according to all current modelling paradigms.

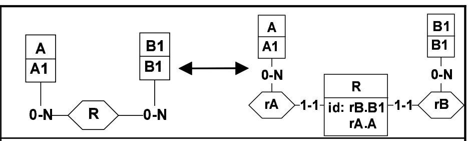
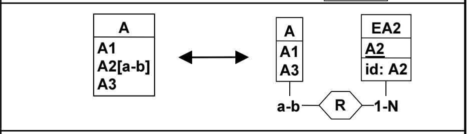
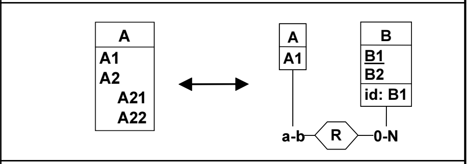
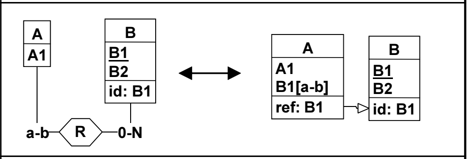
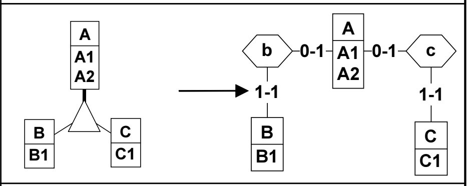
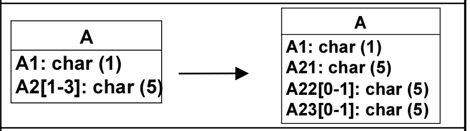
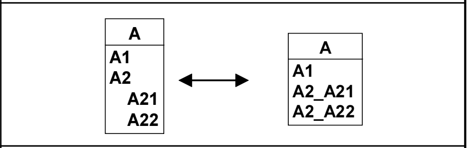
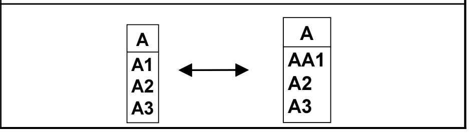
RT-ET: Transforming a relationship type into an entity type. *Inverse:* **ET-RT**	
Att-ET/val: Transforming an attribute into an entity type (value representation). *Inverse:* **ET-Att**	
Disagg: Disaggregating a compound attribute *Inverse:* **Aggreg**	
RT-FK: Transforming a binary relationship type into a foreign key. *Inverse:* **FK-RT**	
ISA-RT: Materializing an ISA relationship type. *Inverse:* **RT-ISA**	
MultiAtt-Serial: Replacing a multivaluated attribute with a series of single-valued attributes that represents its instances. *Inverse:* **Serial-MultiAtt**	
CompAtt-Serial: Replacing a compound attribute with a series of atomic attributes that represent its component attributes. *Inverse:* **Serial-CompAtt**	
Rename-ET or **RenameAtt:** an entity type or an attribute is renamed. *Inverse:* **Rename-ET** or **Rename-Att**	

Figure 2. Major generic schema transformations with their inverse. Entity type and attribute names as well as cardinalities a,b,c, d must be replaced with actual values.

3 In [15], we show how XML structures can be represented in terms of the GER.

4 In [7], a proof system has been developed to evaluate the reversibility of a transformation.

3.2.3. Structural analysis of schema transformations. A transformation is known to replace construct `C` with construct `C′` in schema `S`, to yield new schema `S′`. The effect of a transformation `T` in schema `S` can be specified as follows. We define a schema `S` as a set of constructs. Therefore, set-theoretic relations and operators apply on schemas. Let us consider the structural functions C_-, C_+ and C_0:

- C_- returns the constructs of `S` that have disappeared in `S′`;
- C_+ returns the new constructs that appear in `S′`;
- C_0 returns the constructs of `S` that are concerned by `T`, but that are preserved by transformation (the catalytic constructs of `T`).

3.2.4. Transformation sequence. A transformation sequence is a list of `n` primitive transformations: `S1-to-S2 = (T1 T2 ... Tn)`. For instance, the application of `S1-to-S2 = (T1 T2)` on a schema `S1` consists of the application of `T2` on the schema that results from the application of `T1`, so that we obtain `S2`.

As for schema transformation, a transformation can be inverted. The inverse sequence `S2-to-S1` can be derived from the sequence `S1-to-S1` and can be defined as follows: if `S1-to-S2 = (T1 T2 ... Tn)` then `S2-to-S1` = ($\mathtt{Tn}^{-1}$ ... $\mathtt{T2}^{-1}$ $\mathtt{T1}^{-1}$) *where* $\mathtt{Ti}^{-1}$ is the inverse of `Ti`; and hence `S1 = S2-to-S1(S2)`. In other words, `S2-to-S1` is obtained by replacing each origin schema transformation by its inverse and by reversing the operation order.

The concepts of sequence and its inverse are used for defining the mappings between two schemas. The transformational approach then consists in defining a (reversible) transformation sequence which, applied to the source schema, produces the target schema.

As an illustration, Figure 3 shows a sequence of three transformations often used in database engineering process. The first one (`FK-RT`) replaces a foreign key into a relationship type, the second one (`Serial-CompAtt`) aggregates two attributes and the third one (`Serial-MultAtt`) transform a serie of single-value attributes into a multivalued attribute.

3.2.5. Model translation. A model translation is a particular case of schema conversions [9]. It consists in translating a schema expressed in a data model `Ms` into a schema expressed in another data model `Mt` where `Ms` and `Mt` are two different submodels (i.e., subsets) of GER.

4. Schema and query mapping

In this section, we show how a schema transformation sequence can be used to automatically translate queries between a pair of schemas. More precisely, for a schema transformation sequence between two schemas `S1` and `S2`, we show how this sequence can be used to automatically translate queries posed on `S2` to queries posed on `S1`.

4.1. Model and query language

For simplicity and clarity, we consider a binary model defined as a sub-model of the generic data model described above. This model is compliant with standard files, SQL2 and ER models. It is expressive and generic enough to describe all the main structures and constraints that are explicitly offered by these data models:

- Atomic or compound attributes; single-valued or multivalued attributes;
- Reference, identifier and access groups;
- Entity types with at least one attribute and one identifier;
- Binary, non cyclic relationship types, without attribute;
- ISA relations.

We provide a simple query language based on this binary model: a query (named `Query` here below) is a conjunction of schema constructs. A query answer is a set instances of schema constructs. Any query `Query` over a schema `S` is an expression whose variables are constructs of `S`. The syntax of a query is:

Query `::= Construct | Predicate | [and, Query, Query {, Query}] | [or, Query, Query {, Query}] | [not, Query]`

Predicate `::= [eq, Atom, Atom] | [less, Atom, Atom]`

`Construct` identifies a schema construct being added or deleted by a transformation. In other words, this is one of the constructs that take part in the definition of a schema transformation signature. `Construct` includes variable(s) used to instantiate instances of the construct and it takes one of the forms presented in Table 1. The underscore character is an anonymous variable. `Atom` represents a variable declared in a schema construct. When `eq` refers two variables of the same query, we can simplify the query and omit this predicate, e.g. we need only to write `[att, Person, Id, EP, 4]` instead of `[and, [att, Person, Id, EP, ID], [eq, [ID, 4]]`. In table 2, we illustrate the `Construct` representations of three constructs of Figure 3.

4.2. Schema transformation and query substitution

Let us assume that a schema `S1` is transformed into a schema `S2` and the queries posed on `S1` have to be translated to queries posed on `S2`. Consider first the case where `S1` is transformed into `S2` by a single primitive transformation `T`. The only cases we need to consider in order to translate a query `Q1` posed on `S1` to an equivalent query

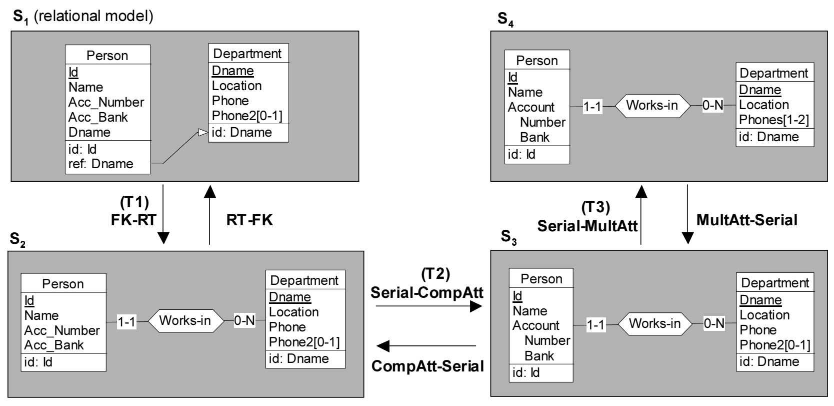


Figure 3. Sequence of schema transformations: a foreign key transformation followed by an aggregation transformation and a transformation of serial attributes into a multi-valued one.

Construct	Syntax	Semantics
Entity type	`[ent, Name, Et]`	represents an entity type called `Name`, and `Et` can be instantiated with instances of `Name`
Attribute	`[att, OwnerName, AttName, Owner, Att]`	represents an attribute `AttName` of a construct `OwnerName`. The type of `OwnerName` can be either an entity type or a compound attribute. `OwnerName` contains the name(s) of the parent(s) of the attribute. `Att` can be instantiate with a value of the attribute associated with the instance `Owner` of `OwnerName`
Relationship type	`[rel, ET1Name, RTName, ET2Name, ET1, ET2]`	represents a relationship `RTName` between entities `ET1Name` and `ET2Name`. `ET1` and `ET2` can be instantiate with entity instances involved in the relationship

Table 1. Syntax and semantics of the main constructs of the generic data model.

Construct	Query Language Representation
Entity type `Person`	`[ent, Person, EP]`
Attribute `Id` of `Person`	`[att, Person, Id, EP, ID]`
Relationship type `Works-in`	`[rel, Person, Works-in, Department, EP, ED]`

Table 2. Examples of construct representation (`EP`, `ED` and `ID` represent variables).

Transformation	Signature	Substitution
`RenameET`	`(name')` ← `RenameET(name)`	`Q2 = [name'/name] Q1`
`RenameAtt`	`(name')`← `RenameAtt(ET,name)`	`Q2 = [ET,name'/ET,name] Q1`
Other	`(S2)`← `T(S1)`	`Q2 = [C`$_{-}$`(T)/query] Q1`
Other (inverse)	`(S2)`← `T`$^{-1}$`(S1)`	`Q1 = [C`$_{-}$`(T`$^{-1}$`)/query] Q2`

Table 3. Schema transformation and query substitution.

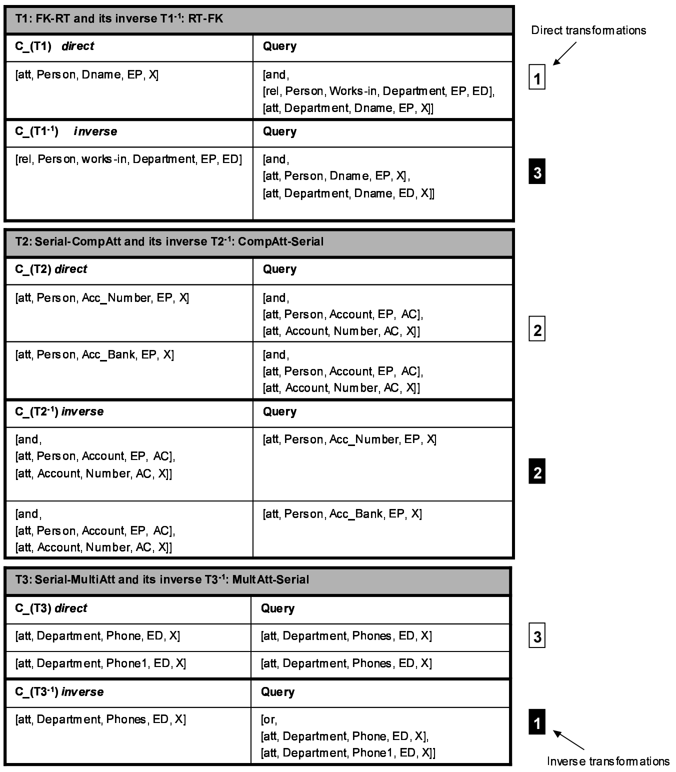
`Q2` on `S2` are to apply renamings and to substitute occurrences of constructs of `C`$_{-}$`(T)` (Table 3). For transformation sequences, the substitutions are successively applied in order to obtain the final query `Q2`.

4.3. Illustration

We illustrate these notions by giving examples of query and update translation between two schemas. We consider the primitive schema transformations `T1`, `T2` and `T3` and their inverse between the pair of schemas `S1` and `S4` illustrated in Figure 3. `T1`, `T2`, `T3` and their inverse are defined in Figure 4 below by means of: (1) their name (2) their structural function `C`$_{-}$ expressed in the `schema` form; and (3) the queries `query` that state how the extents of each constructs of `C`$_{-}$ can be recovered from the extents of the remaining schema constructs `C'`.

For any query on `S4`, the table of Figure 4 can be used to translate constructs of `S4` (the wrapper schema) into ones on `S1` (the database schema), resulting in query on `S1`.

Translation of a query from `S4` to `S1` (Figure 5). ”Find the persons that are reachable via phone number 040-303030” is translated into equivalent queries in `S4` and in `S1` by applying the substitutions ❶ and ❸.


Direct transformations
1
3
2
2
3
1
Inverse transformations

T1: FK-RT and its inverse T1^{-1}: RT-FK	
C_(T1) ***direct***	**Query**
[att, Person, Dname, EP, X]	[and, [rel, Person, Works-in, Department, EP, ED], [att, Department, Dname, EP, X]]
C_(T1^{-1}) ***inverse***	**Query**
[rel, Person, works-in, Department, EP, ED]	[and, [att, Person, Dname, EP, X], [att, Department, Dname, ED, X]]

T2: Serial-CompAtt and its inverse T2^{-1}: CompAtt-Serial	
C_(T2) ***direct***	**Query**
[att, Person, Acc_Number, EP, X]	[and, [att, Person, Account, EP, AC], [att, Account, Number, AC, X]]
[att, Person, Acc_Bank, EP, X]	[and, [att, Person, Account, EP, AC], [att, Account, Number, AC, X]]
C_(T2^{-1}) ***inverse***	**Query**
[and, [att, Person, Account, EP, AC], [att, Account, Number, AC, X]]	[att, Person, Acc_Number, EP, X]
[and, [att, Person, Account, EP, AC], [att, Account, Number, AC, X]]	[att, Person, Acc_Bank, EP, X]

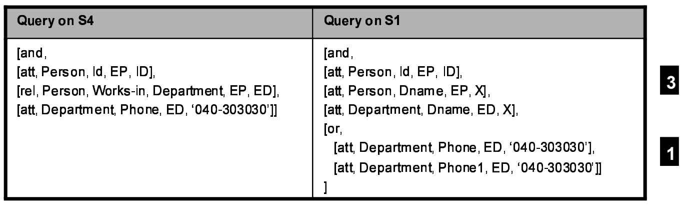
T3: Serial-MultiAtt and its inverse T3^{-1}: MultAtt-Serial	
C_(T3) ***direct***	**Query**
[att, Department, Phone, ED, X]	[att, Department, Phones, ED, X]
[att, Department, Phone1, ED, X]	[att, Department, Phones, ED, X]
C_(T3^{-1}) ***inverse***	**Query**
[att, Department, Phones, ED, X]	[or, [att, Department, Phone, ED, X], [att, Department, Phone1, ED, X]]

Figure 4. Example of schema transformations and the queries that state how the extents of each construct of `C_` can be recovered from the extents of the remaining schema constructs `C'(T)`.

5. Wrapper development

Since the mapping between wrapper and database schemas is formally defined, we can expect them to be a sound basis to build the wrapper in a systematic way. Indeed, while the structural mapping `T` of a transformation defines a rewriting rule that can be used to transform the input query, its instance mapping `t` states how the instance of the target construct can be derived from that of the source construct. Therefore, these mappings can be used to define the query translation logic and the data transformation rules of the wrapper that implements this transformation. This analysis is still valid for transformation sequences, so that complete wrappers can be formally specified by such sequences.

Each wrapper is developed as a program component dedicated to a specific database model and to a specific database. It comprises two parts, namely a *model layer*, in which the aspects specific to a given data model (e.g., RDB



Query on S4	Query on S1
[and, [att, Person, Id, EP, ID], [rel, Person, Works-in, Department, EP, ED], [att, Department, Phone, ED, '040-303030']]	[and, [att, Person, Id, EP, ID], [att, Person, Dname, EP, X], [att, Department, Dname, ED, X], [or, [att, Department, Phone, ED, '040-303030'], [att, Department, Phone1, ED, '040-303030']]]

Figure 5. Example of a query translation from S4 to S1.

or standard files) are coped with, and a *database layer* that is dedicated to the specific database schema. While the model layer is common to all the databases built in this model, the wrapper/database schemas mapping is hardcoded rather than interpreted from mapping tables as it is the case in other approaches.

In this section, we will discuss the baselines of this approach.

5.1. Schema and mapping definition

5.1.1. Model translation. In our generic approach, model translation is defined as a model-driven transformation within the generic data model defined in Section 3. A model-driven transformation applies on a schema. It can be defined by `m(Ms, Mt)` where `Ms` and `Mt` are two different submodels, i.e., subsets of the generic data model. It consists in applying the relevant transformations on the relevant constructs of the schema expressed in `Ms` in such a way that the final result complies with `Mt`. A model-driven transformation is expressed as a transformation plan made up of a sequence of `<condition, action>` statements and control structures, where condition is a structural predicate and action is a transformation. The meaning is obvious: apply action `action` on each construct that satisfies the predicate `condition`.

As an illustration of model translation, we consider the simplified transformation plan between the relational model and the ER model (Figure 6). This transformation plan can be applied to any schema expressed in the relational model (for instance, schema `S1` of Figure 3). Its execution produces two result types: (1) a target schema expressed in the ER model and equivalent to the source schema (in our example, the resulting target schema is `S2`); and (2) a schema transformation sequence that reports all the transformations applied by the transformation plan (in our example, the schema transformation sequence is made up of only one transformation `FK-RT` on `Dname` of `Person`).

5.1.2. Refinement transformations. Model translation provides automated mechanisms that consist in trans-

```
1- For each foreign key F of an entity type $ET_s$ that references another entity type $ET_t$ do:
      apply FK-RT to F;
2- For each entity type E, do:
      If E meets the precondition of ET-RT, apply ET-RT to E;
3- For each entity type E, do:
      if E meets the precondition of ET-Att, apply ET-Att to E;
4- For each RT relationship type R, do:
      if R meets the precondition of RT-ISA, apply RT-ISA to R;
```

Figure 6. Transformation plan between the relational model and the ER model (simplified version).

lating each construct of the source database into the closest constructs of another data model without attempting any semantic interpretation. It only captures the structure of the database schema and largely ignores the hidden semantic constructs. However, weakness of available database models and information hiding programming practices lead to incompleteness of database schemas that only contains the structures *explicitly* expressed in the DDL code. For example, in Figure 3, the schema `S1`, being expressed according to the relational model, exhibits a sequence of attributes (`Acc_Number` and `Acc_Bank` of `Person`) that seemingly are originated from a compound attribute which was decomposed in order to comply with the relational model. Another frequent example of hidden construct elicitation is the recovery of foreign keys that were not explicitly declared in legacy relational databases based on, say, old version of Oracle or Sybase.

This process of semantic interpretation consists in identifying and extracting all the relevant concepts underlying a database schema. To accomplish this, we build on a proven approach, namely the *DB-MAIN reverse engineering methodology* [6]. This approach has been already integrated in the wrapper development methodology presented in [16] and [17] and will not be discussed further in this paper. Its key feature is twofold. First all the schemas, whatever their modelling language, are expressed in the GER. Secondly, it uses the same transformational approach than that of this paper.

5.2. Wrapper generation support

The wrapper generation is supported by the DB-MAIN tool, a general-purpose database engineering CASE and meta-CASE environment that offers sophisticated database application engineering toolsets. DB-MAIN includes advanced processors such as DDL parsers, transformation toolboxes, reverse engineering processors and schema analysis tools. In particular, DB-MAIN offers a rich set of transformational operators (including semantics-preserving ones) that offers a rich set of transformational operators (including semantics-preserving ones) that allow developers to define mappings in a systematic and formalized, though intuitive way. Another interesting feature of DB-MAIN is the meta-CASE layer, which allows method engineers to customize the tool and to add new concepts, functions, models and even new methods. In particular, DB-MAIN offers a complete development language, *Voyager 2*, through which new functions and processors can be developed and seamlessly integrated into the tool. Further details on DB-MAIN can be found in [8]. In the limited scope of this paper, we describe the two main *Voyager 2* programs dedicated to the wrapper code generation.

History analyzer. DB-MAIN automatically generates and maintains a history log of all the transformations that are applied when the developer carries out any engineering process such as wrapper schema definition. This history is completely formalized in such a way that it can be replayed, analyzed and transformed. An history basically is a procedural description of inter-schema mappings. The history analyzer parses history logs and transforms them into non-procedural annotations that define the inter-schema object mappings.

Wrapper encoders. The wrappers are automatically generated from the mapping annotations. Two wrapper interfaces are provided, namely SQL-based through a variant of JDBC, and object-based. At the current time, wrapper encoders for COBOL files and relational data structures are available.

6. Experiment

The approach described in this paper has been applied on several actual systems; two of them are briefly described in this section. The first application (`A-COB`) is a small size COBOL test bed we have developed to precisely check the various versions of our generator. It includes a 3-file database that comprises examples of complex hidden structures and constraints, together with a 400-LOC application programs. The second application (`B-RDB`) is a (collection of similar) INFORMIX relational database(s) dedicated to taxes management in a Belgian municipality. Wrappers of application `B-RDB` have been integrated into a database federation controlled through a light mediator developed in JAVA/HTML. The latter provided some functions to arbitrate among conflicting data from the taxes databases.

We have also migrated each of them in the other technology, which was a straightforward process, since both databases comprises flat files/tables only. This provides us with two additional case studies, namely `A-RDB` and

B-COB. According to the architecture described in [16] and recalled in this paper, the size of a wrapper is the sum of the LOC of the model layer and of that of the database layer. The first layer has a constant size, which is, for the current version of the generators, of 7,500 LOC for RDB wrappers and 4,400 LOC for COBOL wrappers. Evaluating the cost of the database layer is more complex, since it depends of several factors:

- the underlying DBMS on which the wrapper is dedicated;
- the size of the database and wrapper schemas;
- the number and the type of schema transformations of the sequence.

Table 4 specifies, for applications A and B, the composition of the wrapper schema (number of entity types and attributes), the number of transformations that define the mappings and the number of implicit constructs that have been elicited. Note that, due to the simplicity of the database schemas (flat structures only), these figures are valid for both COBOL and SQL technologies, and therefore apply to the four case studies.

Application	Wrapper Schema Size	Transformation Sequence Size	Implicit Constructs
A	3 entity types 15 attributes	3 transformations	2 implicit structures
B	30 entity types 120 attributes	60 transformations	35 implicit structures

Table 4. The two case studies and their size.

Table 5 gives the size of the code fragment (COBOL for applications A and C for applications B) that is generated for each of the following constructs of the wrapper schema: entity type, attribute, implicit compound attribute and implicit multivalued attribute. The table should be augmented with the score of additional constructs for other case studies.

Explicit constructs	COBOL Wrapper	RDBMS Wrapper
Entity Type	380	125
Attribute	120	40
Implicit Compound Attribute	150	40
Implicit Multivalued Attribute	220	55

Table 5. LOC size of explicit or implicit constructs.

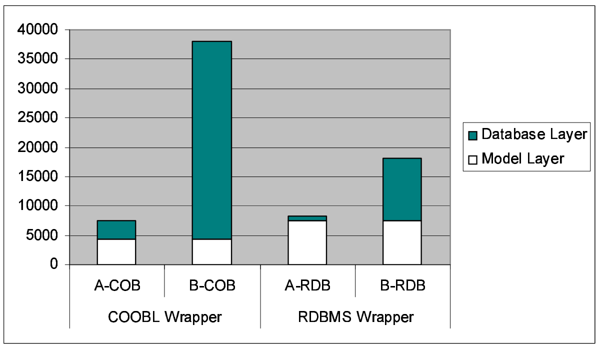


Figure 7. LOC sizes of model and database layers of wrappers of the four case studies.

The size of the wrappers can be computed from these tables. They are illustrated in Figure 7. The differences between both technologies stem from the way the database layers are generated. In COBOL wrappers, each construct of the wrapper schema is managed by a specific code fragment (in some sense, the schema is hard-coded in the wrapper). In RDB wrappers, thanks to the use of the ODBC interface, the schema is described by a table, so that each schema construct requires much less LOC than the COBOL approach.

7. Conclusions

Data wrapping is one of the most powerful techniques to bridge existing (most often legacy) databases with modern architectures. Its main goal is model conversion, that addresses two inverse streams, namely query translation and data transformation.

In this paper, we have focused on the query translation process in database wrappers. This process relies on a special kind of inter-schema mapping, namely sequences of transformations. By replacing the schemas constructs names in the wrapper query with their database equivalent, we produce a database query that can be executed on the actual data. This systematic approach can be automated, in such a way that wrappers can be generated based on the schema mappings. A specific plug-in has been developed for the DB-MAIN CASE tool. Considering two schemas and their mappings, expressed by a sequence of transformations, it generates wrappers for COBOL files and relational databases. The approach and the tool have been applied, among others, for building federated databases mixing both legacy and modern technologies.

One problem we encountered when building wrappers is coping with non standard constructs. Indeed, the generic model, despite its power, cannot express all integrity constraints. When a specific constraint is found in the reverse engineering process, it is expressed as a generic constraint, described by a free text annotation. The wrapper generator includes in the wrapper code a skeleton that documents the constraint. It is up to the programmer to write the specific code for this constraint. Therefore, our approach can be qualified semi-automatic. The technology we have developed is being integrated into a development environment for business-to-customer applications that are built on top of legacy databases.

References

[1] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vinci. Retrieving and integrating data for multiple sources: the momis approach. *Data and Knowledge Engineering*, 36, 2001.

[2] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing object-relational data as XML. In *WebDB (Informal Proceedings)*, pages 105–110, 2000.

[3] M. Fernandez, W. Tan, and D. Suciu. Silkroute: Trading between relations and XML. In *Proceedings of the Ninth International World Wide Web Conference*, 2000.

[4] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[5] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[6] J.-L. Hainaut. Introduction to database reverse engineering. Technical report, University of Namur, 2002.

[7] J.-L. Hainaut. *Transformation of Knowledge, Information and Data: Theory and Applications*, chapter Transformation-based Database Engineering. IDEA Group, 2005.

[8] J.-M. Hick, V. Englebert, J. Henrard, D. Roland, and J.-L. Hainaut. The DB-MAIN database engineering CASE tool (version 6.5) - functions overview. Db-main technical manual, Institut d'informatique, University of Namur, 2002.

[9] D. Lee, M. Mani, F. Chiu, and W. W. Chu. NeT and CoT: Translating relational schemas to XML schemas using semantic constraints. In *ACM International Conference on Information and Knowledge Management*, 2002.

[10] I. Manolescu, D. Florescu, and D. K. Kossmann. Answering {XML} queries over heterogeneous data sources. In *VLDB*, pages 241–250, 2001.

[11] P. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications - a schema transformation approach. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 96–113, 1999.

[12] P. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures. In *CAiSE Proceedings*. Springer-Verlag, 2002.

[13] J. Shanmugasundaram, J. Kiernan, E. J. Shekita, C. Fan, and J. Funderburk. Querying XML views of relational data. In *Proceedings of the 27th VLDB Conference*, 2001.

[14] P. Thiran. *Legacy Database Federation - A Combined Reverse-Forward Approach*. Phd thesis, University of Namur, October 2003.

[15] P. Thiran, F. Estievenart, J.-L. Hainaut, and G.-J. Houben. Exporting databases in XML a conceptual and generic approach. *Proc. of CAiSE Workshops (WISM04)*, 2004.

[16] P. Thiran and J.-L. Hainaut. Wrapper development for legacy data reuse. In I. C. Press, editor, *WCRE Proceedings*, 2001.

[17] P. Thiran, G.-J. Houben, J.-L. Hainaut, and D. Benslimane. Updating legacy databases through wrappers: Data consistency management. In I. C. Press, editor, *WCRE Proceedings*, 2004.