Automatic Generation of Model Translations

Paolo Papotti and Riccardo Torlone

Università Roma Tre
Roma, Italy
{papotti,torlone}@dia.uniroma3.it

Abstract. The translation of information between heterogeneous representations is a long standing issue. With the large spreading of cooperative applications fostered by the advent of the Internet the problem has gained more and more attention but there are still few and partial solutions. In general, given an information source, different translations can be defined for the same target model. In this work, we first identify general properties that "good" translations should fulfill. We then propose novel techniques for the automatic generation of model translations. A translation is obtained by combining a set of basic transformations and the above properties are verified locally (at the transformation level) and globally (at the translation level) without resorting to an exhaustive search. These techniques have been implemented in a tool for the management of heterogeneous data models and some experimental results support the effectiveness and the efficiency of the approach.

1 Introduction

1.1 Goal and Motivations

In today's world of communication, information needs to be shared and exchanged continuously but organizations collect, store, and process data differently, making this fundamental process difficult and time-consuming. There is therefore a compelling need for effective methodologies and flexible tools supporting the management of heterogeneous data and the automatic translations from one system to another.

In this scenario, we are involved into a large research project at Roma Tre University whose goal is the development of a tool supporting the complex tasks related to the translation of data described according to a large variety of formats and data models [1,2,3]. These include the majority of the formats used to represent data in current applications: semi-structured models, schema languages for XML, specific formats for, e.g., scientific data, as well as database and conceptual data models. In this paper, we focus our attention on the problem of the automatic generation of "good" data model translations.

We start observing that, in general, given a data source, different translations can be defined for the same target model. To clarify this aspect, let us consider the example in Figure 1 where the relational schema a is translated into an XML

© Springer-Verlag Berlin Heidelberg 2007

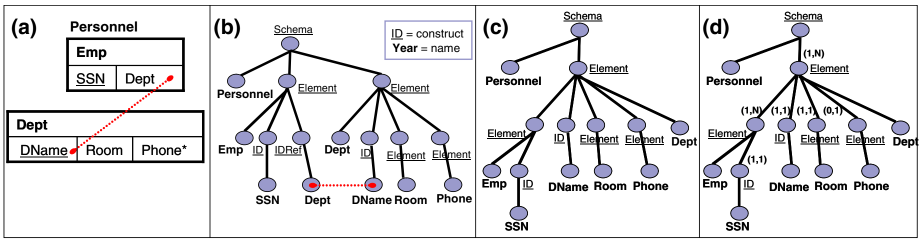


Fig. 1. The translation of a relational schema into an XML based model

based structure. Actually, several solutions are possible since it is well known that different strategies can be followed [4]. We report just three of them.

In our example, we can choose between a nested-based (schemas *c* and *d*) and a flat-based (schema *b*) structure. The latter can be easily generated, but the schema we obtain is probably not desirable in a model with nesting capabilities. Moreover, the question arises whether we want to force "model" constraints like the presence of an order or the absence of duplicates. The second point is that differences between translations are not only structural. For instance, schemas *c* and *d* are similar but they have significant differences in the schema semantics, since *d* also includes cardinality constraints on the elements. Finally, the efficiency of the translation is clearly an issue [5,6].

In order to tackle this problem, in this paper we first identify general properties that the translations should fulfill and investigate the conditions under which a translation can be considered better than another. We then propose efficient methods for the automatic generation of schema and data translations from one model to another. We also show experimental results obtained with a tool for the management of heterogeneous data models in which the proposed methods have been implemented.

1.2 Related Works and Organization

The problem of model translation is one of the issues that arises when there is the need to combine heterogeneous sources of information. Many studies can be found on this problem. For instance, translations between specific pairs of models have been deeply investigated [4,7] and are widely supported in commercial products. Our goal is more general: the development of a flexible framework able to automatically translate between data models that, in principle, are not fixed a priori. In recent years, an aspect of the translation problem that has been deeply studied is data exchange [8], where the focus is on the translation of data between two fixed schemas, given a set of correspondences between the elements. Recently, the problem has been set in the general framework of *model management* [9], where a set of generic operators is introduced to cope, in a uniform way, various metadata related problems. One of them is the ModelGen operator that corresponds to the problem tackled in this paper. An early approach to

ModelGen for conceptual data models was proposed by Atzeni and Torlone [2] with a tool based on an internal metamodel and a library of transformations. Following works [3] and similar approaches [10,19] has been presented in the last years. Currently, there are two active projects working on this subject. Atzeni et. al [1] recently provided a comprehensive solution based on a relational dictionary of schemas, models and translation rules. Their approach however does not consider the automatic generation of translations. The approach of Bernstein et al. [11] is also rule-based and it introduces incremental regeneration of instance mappings when source schema changes. A detailed description of their approach has not yet appeared. Our contribution is orthogonal to both projects. In previous works of ours [3] we have focused our attention to the management of Web information and we have proposed a general methodology for the translation of schema and data between data models. In this paper, the focus is on the *automatic generation* of translations based on the *ranking* of the possible solutions. MOF [12] is an industry-standard framework where models can be exchanged and transformed between different formats and provides a uniform syntax for model transformation. Our approach is complementary: we provide methods to automatically perform translations between models, possibly expressed in a MOF-compliant way.

The rest of the paper is organized as follows. In Section 2 we provide the needed background and, in Section 3, we investigate the general properties of model translations. In Section 4 we present the algorithms for the automatic generation of translation and, in Section 5, we provide some experimental results. Finally, in Section 6, some conclusions are drawn and future work is sketched.

2 Background

2.1 Translations, Metamodel and Patterns

We identify four levels of abstraction: (1) *data* (or *instances*) organized according to a variety of (semi) structured formats (relational tables, XML documents, HTML files, scientific data, and so on); (2) *schemas*, which describe the structure of the instances (a relational schema, a DTD, an XML Schema or one of its dialects, etc.); (3) *(data) models*, that is, formalisms for the definition of schemas (e.g., the relational model, the XML Schema model or a conceptual model like the ER model), and (4) a *metamodel*, that is, a general formalism for the definition of models.[1]

In this framework, a *schema translation* from a source model M_s to a target model M_t is a function $\sigma : \mathcal{S}(M_s) \rightarrow \mathcal{S}(M_t)$, where $\mathcal{S}(M)$ denotes the set of schemas of M, and $\mathcal{I}(S)$ the set of instances of S. If $S \in \mathcal{S}(M_s)$ then $\sigma(S)$ is called the $\sigma-$*translation* of S (this corresponds to the ModelGen operator [9]). Similarly, a *data translation* from a source schema S_s to a target schema S_t is a function $\delta : \mathcal{I}(S_s) \rightarrow \mathcal{I}(S_t)$.

[1] We refer to a "database" terminology; in other works (e.g., [9,12]), a schema is called *model*, a model is called *metamodel*, and a metamodel is called *metametamodel*.

As others [1,2,3,10,11], our approach is based on a unifying *metamodel* made of a set of *metaprimitives* each of which captures similar constructs of different data models. More precisely, a metaprimitive represents a set of constructs that implement, in different data models, the same basic abstraction principle [14]. For instance, a set of objects is represented by a class in ODL and by an entity in the Entity-Relationship model. Clearly, metaprimitives can be combined. We will call a specific combination of metaprimitives a *pattern*. In this framework, a model is defined by means of: (i) set of *primitives*, each of which is classified according to a metaprimitive of the metamodel, and (ii) a set of patterns over the given primitives.

As an example, the table in Figure 2 describes a set of models. The first column contains a set of possible patterns over the metaprimitives of the metamodel. Each pattern has a metaprimitive m as root and a collection of metaprimitives that are used as components of m ("*" means 0 or more times). In the other columns of the table different models are defined by listing the patterns used and the names given to them in the model. For instance, the relational model is defined by means of a set having the table pattern (which correspond to the metaprimitive relation) which is composed by a number of attribute constructs, one key and, possibly, a foreign key.

	XmlSchema	**DTD**	**ODL**	**Relational**	**ER**
Element	*element*	*element/entity*	-	-	-
Domain	√	√	-	-	-
AttributeOfElement*	√	√	-	-	-
Key	√	√	-	-	-
Cardinality	√	√	-	-	-
Object	-	-	*class*	-	*entity*
Key	-	-	√	-	√
Attribute*	-	-	√	-	√
Relationship	-	-	√	-	√
Relation	-	-	-	*table*	-
Attribute*	-	-	-	√	-
Key	-	-	-	√	-
Foreign key	-	-	-	√	-
Domain	*type*	*type*	*type*	*type*	*type*
Struct*	-	-	√	-	-
Restriction	√	-	√	√	√
List	√	√	-	-	-
Extension	√	-	-	-	-
Key	*key*	*key*	*key*	*key*	*key*
Domain	√	√	√	√	√
Attribute	-	-	*attribute*	*attribute*	*attribute*
Domain	-	-	√	√	√
ForeignKey	-	-	-	√	-
Cardinality	-	-	√	-	√
Relationship	-	-	*relationship*	-	*relationship*
Cardinality*	-	-	-	-	√
AttributeOfObject	-	-	-	-	√
RelationshipType	-	-	√	-	-
...					

Fig. 2. A set of models described by patterns

A pattern corresponds to a context free grammar that makes use of an alphabet denoting the primitives of the metamodel. We call a string of this grammar a *structure*. A *schema* can be obtained by associating names to the symbols of a structure. For instance, the schema (b) in Figure 1 is obtained by adding the nodes in bold to the rest of the tree, which corresponds to the underlying structure.

2.2 A Transformational Approach

In [3], we have introduced a general methodology for model translation based on three main steps: (1) the source schema S is first represented in terms of the metamodel so that it can be compared with the target data model definition; (2) source schema and target model may share some constructs (metaprimitives), but others must be translated or eliminated to obtain a schema consistent with the target data model. This operation is performed on S: the system tries to translate the metaprimitives of S into metaprimitives of the target model or, if the translation fail, it removes them; (3) a rewriting of the generated schema in terms of the target model syntax is executed.

The translation step, which is the fundamental phase of the process, takes as input a schema expressed in terms of (patterns of) metaprimitives. As the number of metaprimitives is limited, it is possible to define a library of basic and "generic" transformations that can be composed to build more complex translations. These basic transformations implement rather standard translations between metaprimitives (e.g., from a relation to an element or from a n-ary aggregation to a binary one). Representatives of such transformation have been illustrated in [3].

Actually, each basic transformation b has two components: a schema translation σ and a data translation δ. Its behavior can be conveniently represented by a *signature* $b[\mathbf{P}_{in}:\mathbf{P}_{out}]$, that is, an abstract description of the set of patterns $\mathbf{P}_{in}$ on which p operates and of the set of patterns $\mathbf{P}_{out}$ introduced by p. Note that this description makes the approach independent of the actual implementation of the various transformations. As an example, the signature of an unnesting transformation that transforms each nested element into a set of flat elements related by foreign keys is the following:

$$\begin{aligned}&b[\{\mathit{ComplexElement}(\mathit{ComplexElement}+, \mathit{AtomicElement}*, \mathit{Domain})\}: \\ &\quad \{\mathit{ComplexElement}(\mathit{Key}+, \mathit{ForeignKey}*, \mathit{AtomicElement}*, \mathit{Domain})\}]\end{aligned}$$

It turns out that the effect of a transformation with signature $b[\mathbf{P}_{in}:\mathbf{P}_{out}]$ over a structure that makes use of a set of primitives $\mathbf{P}$ is a structure using the primitives $(\mathbf{P} - \mathbf{P}_{in}) \bigcup \mathbf{P}_{out}$.

3 Transformations and Translation

In this section, we first investigate general properties of basic transformations and then introduce properties for complex translations.

3.1 Properties of the Basic Transformations

Several properties characterizing the correctness, the effectiveness and the efficiency of a basic transformation can be defined, and this issue has been largely debated in the literature (see for instance [2]). Among them, we have focused our attention into the properties that follow.

The first property states a consistency relationship between the schema translation and the data translation which compose a basic transformation.

Definition 1. *A basic transformation $b = (\sigma, \delta)$ is* consistent *if for each schema $S \in \mathcal{S}(M_s)$ and for each instance I of S, $\delta(I)$ is an instance of $\sigma(S)$.*

A key aspect for a schema transformation is its "correctness", that is, the fact that the output schema is somehow equivalent to the input one. The equivalence of two schemas is a widely debated topic in literature, and all the approaches rely on the ability of the target schema to represent the same information of the source one [15,16,17]. In other words, all data associated with the input schema can be recovered from the data associated with the output schema. This notion has been named *equivalence preserving* or *information preserving* and have been formalized by means of the following properties:

- a data translation δ from S_s to S_t is *query preserving* w.r.t. a query language $\mathcal{L}$ if there exists a computable function $F : \mathcal{L} \rightarrow \mathcal{L}$ such that for any query $Q \in \mathcal{L}$ over S_s and any $I \in \mathcal{I}(S_s)$, $Q(I) = F(Q)(\delta(I))$;
- a data translation δ from S_s to S_t is *invertible* if there exists an inverse δ^{-1} of δ such that, for each instance $I \in \mathcal{I}(S_s)$, $\delta^{-1}(\delta(I)) = I$.

In our context, the property that actually guarantees the equivalence depends on the internal model used to represent the schemas. In [1] the internal model is based on a relational dictionary, and it has been shown that calculus dominance and query dominance are equivalent for relational settings [16]. In contrast, invertibility and query preservation do not necessarily coincide for XML mappings and query languages [18,15]. In the following, we will refer to a notion of "equivalence preserving" that relies on query preservation.

As we have shown in the introduction, even if we assume that all the transformations preserve the above properties, there are transformations that are preferable than others. Different issues can be considered in this respect: redundancy, ease of update maintenance, ease of query processing with respect to certain workload, and so on. We therefore assume that a preference relationship can be defined over the basic transformations according to one or more of these aspects. One important point is that this preference relationship depends, in many cases, on the target model. For instance, a translation to an object model (with both relationships and generalizations) that is able to identify generalization hierarchies between classes is preferable to a translation that only identifies generic relationships between them. This is not true if the target is the relational model.

First of all, we say that two basic transformations $b[\mathbf{P}_{in} : \mathbf{P}_{out}]$ and $b'[\mathbf{P}'_{in} : \mathbf{P}'_{out}]$ are *comparable* if either $\mathbf{P}_{in} \subseteq \mathbf{P}'_{in}$ or $\mathbf{P}'_{in} \subseteq \mathbf{P}_{in}$.

Definition 2. *Given a set L of basic transformations and a target data model M_t a* preference relationship $>_{M_t}$ towards M_t *is a poset over comparable transformations in L. Given two comparable basic transformations b_1 and b_2 in L, we say that b_1 is* preferable *to b_2 w.r.t. M_t if $b_1 >_{M_t} b_2$.*

Example 1. Let P_1 be a pattern denoting an Entity, P_2 be a pattern denoting a Relationship, and P_3 a pattern denoting a Generalization. Given the basic transformations b_1 with signature $[\{P_3\}:\{P_1, P_2\}]$, which translates generalizations into entities and relationships, and $b_2[\{P_3\}:\{P_1\}]$, which simply translates generalizations into entities. Then, we can state that $b_1 >_{M_t} b_2$ if M_t is a model with entities and relationships. The rationale under this assertion is that b_1 takes more advantage than b_2 of the *expressiveness* of the target data model.

In Section5, we will concretely specify a specific preference relationship that is suitable for our purposes.

We finally define a property for the evaluation of the performance of a basic transformation. The best way to measure the effective cost of a transformation would be the evaluation of its execution at runtime. Obviously we would prefer to not actually execute basic transformations on instances to compare their performance. Since execution time optimization is not our primary goal, an estimation of each basic transformation complexity is a reasonable solution. In particular, we assume that the designer provides a specification of the complexity of the algorithm with respect to the size of the database. For instance, the complexity of the unnesting transformation described in the previous is linear with respect to the database. We denote the complexity for a basic transformation b_x with $c(b_x)$.

Definition 3. *Given a set L of basic transformations an* efficiency relationship *$\succ$ is a total order over L such that $b_i \succ b_j$ if $c(b_i) > c(b_j)$. Given two basic transformations b_i and b_j, we say that b_i is more* efficient *than b_j if $b_j \succ b_i$.*

3.2 Properties of Translations

We have just defined some *local* properties of transformations, but we would like to study also *global* properties of entire translations.

It has been observed that, in the transformational approach, if every transformation b_i in the library is equivalence preserving, the information preservation for the sequential application of two or more transformations is guaranteed by construction [2,17]. The same guarantee applies for the schema validation: the generated output schema cannot contain primitives that are not allowed in the target data model. It turns out that a translation $t = b_1, \ldots, b_n$ from M_s to M_t is *consistent* if each b_i in σ is consistent.

We now extend the preference property to translations.

Definition 4. *Given two translations $t = b_1, \ldots, b_m$ and $t' = b'_1, \ldots, b'_n$ and a target model M_t, $t >_{M_t} t'$ if: (i) there exists a transformation b_i in t such that $b_i >_{M_t} b'_j$, for some transformation b'_j in t', and (ii) there is no transformation b'_k in t' such that $b'_k >_{M_t} b_l$ for some transformation b_l in t.*

It is easy to show that the above relationship is a poset over the set of all possible translations.

Example 2. Assume that L contains three basic transformations: b_1 with signature $[\{P_3\}$:$\{P_1, P_2\}]$, which translates generalizations (P_3) in entities and relationships (P_1 and P_2 respectively), b_2 with $[\{P_1, P_2\}$:$\{P4\}]$, which translates entities and relationships into elements (P_4), and $b_3[\{P_3\}$:$\{P_1\}]$, which translates the generalizations into entities. Consider a target data model M_t with just the element pattern (a subset of DTD). If we assume the preference discussed in Example 1 ($b_1 >_{M_t} b_3$), and consider the following translations: $t_1 = b_3, b_2$, $t_2 = b_2, b_3$, and $t_3 = b_1, b_2$, then we have that $t_3 >_{M_t} t_2$ and $t_3 >_{M_t} t_1$.

The efficiency of a translation can be defined with different levels of granularity. In [3] we proposed a preliminary evaluation based on the *length* of the translation, that is, the number of basic transformations that composed the actual solution. We now extended the definition: the efficient solution is the translation t that globally minimize the cost of the basic transformations $b_1, \dots, b_n$ that compose t.

Definition 5. *Given two schema translations $t = b_1, \dots, b_m$ and $t' = b'_1, \dots, b'_n$, t is more* efficient *than t' if* $\max_{j=1}^{m} c(b_j) < \max_{i=1}^{n} c(b'_i)$.

If we consider the data model translation problem (a translation of source schema S into a target model M_t given a library L of basic transformations $\{b_1, \dots, b_n\}$), it turns out that the above properties lead to two different classifications of the possible solutions for the problem based on the orthogonal notions of efficiency and preferability.

Definition 6. *A translation t is* optimal *if there is no other translation that is more preferable and more efficient than t.*

Note that, in general, several optimal translations can exist. We will see in the next section how the classification of translations can be the basis for an automatic ranking of the solutions and for an efficient algorithm that retrieves solutions without generating all the alternatives.

4 Automatic Generation of Translation

In this section we present two approaches to the problem of the automatic generation of translations. The former is based on an exhaustive search, the latter relies on a best-first technique and is much more efficient.

4.1 Computing and Ranking all Translations

In [3], we have proposed a basic strategy that follows a greedy approach: given a source schema S_s, a target model M_t and a library of basic transformations L, this method applies exhaustively the following rule over a working schema S, which initially coincides with S_s, and an initially empty sequence of transformations t.

if (a) the S makes use of a pattern P that is not allowed in the target model, and (b) there exists a transformation $b(\sigma, \delta) \in L$ whose effect is the removal of P and (possibly) the introduction of patterns allowed in the target model, then append b to t and set S to $\sigma(S)$.

When condition (a) fails, the process terminates successfully and the sequence t is a solution for the data translation problem.

This simple method can be extended to an algorithm ComputeAllSolutions that generates all the possible solutions for S_s. It is possible to show that this algorithm is complete in the sense that every valid translation from the S_s to the target data model is in the solution set T. This algorithm is shown in Figure 3.

```
Input: A schema $S_s$, a target model $M_t$ and a library of basic
transformations $L = \{b_1, \ldots, b_n\}$.
Output: A set of all translations $T = \{t_1, \ldots, t_n\}$ of $S_s$ into $M_t$ (each $t_i$ is a
sequence of basic transformations in $L = \{b_1, \ldots, b_n\}$).
begin
(1)    Set $t$ to the empty translation and $st$ to the structure of $S_s$;
(2)    Add $(t, st)$ to the set of possible solution $Sol$;
(3)    while, for each $s \in Sol$, there is a pattern $P$
       in $st$ s.t. $P$ is not allowed in $M_t$ do
(4)    let $B$ denote the set of all $(b_p, t', st'_i)$ branches under $st$ such that:
       (a) $b_p$ is a basic transformation whose input signature matches $P$,
       (b) $t'$ is a copy of the actual $t$,
       (c) $st'_i$ is the resulting structure after $b_p$'s application;
(5)    for each branch $(b_p, t', st'_i)$ in $B$:
(6)        if $b_p \in t'$:
(7)        then discard $(b_p, t', st'_i)$;
(8)        else append $b_p$ to $t'$ and add $(t', st'_i)$ to $Sol$.
       end while
(9)    Add the valid translation $t$ to the solution's set $T$.
(10) Return $T$.
end
```

Fig. 3. ComputeAllSolutions algorithm

Observe that in step (3), we allow the search of a basic transformation to consider any pattern of the input structure st. Since there could be basic transformations that are commutative (the result of the translation could not depend on the order of the basic transformations) we can have solutions that are different but composed by the same basic transformations set. For example, translating to the *DTD* data model, the basic transformation b_x that removes a metaprimitive that cannot be transformed, as the Namespace construct, can be added to the translation at any place in the sequence: $b_i, b_j, b_x \equiv b_x, b_i, b_j \equiv b_i, b_x, b_j$.

We can have translations in T that are equivalent, but notice that we only have translations in which the same transformation does not appear more than once. Consider steps 5-8: if a basic transformation has been already added to t, we discard the branch. This choice prevents also loops in case of pairs of basic transformations that add and remove the same metaprimitive (of a pattern of them).

This algorithm captures all the possible valid translations, including optimal ones. Consider now the efficiency and preferability issues. We can use them to rank the solutions in T and expose to the user only the translations that are in the optimal set. Given the set of solutions $T = \{t_1, \ldots, t_n\}$ we can order T according to $>_{M_t}$ and according to $\succ$: we get two ordered lists of transformations. The optimal set is the union of the top elements in these lists. We show experimentally next that in the optimal set there are often several solutions. This is rather intuitive: some solutions are better in terms of efficiency, others are better in terms of preferability.

Note that, even if the solution's generation and ranking are efficient, an exhaustive exploration of the search space is required. In particular, if the set of basic transformations is large, an exhaustive search can be expensive, since the complexity of the algorithm depends exponentially on the size of the library. Another approach it is introduced next to overcome this limit.

4.2 Best-First Search Algorithm

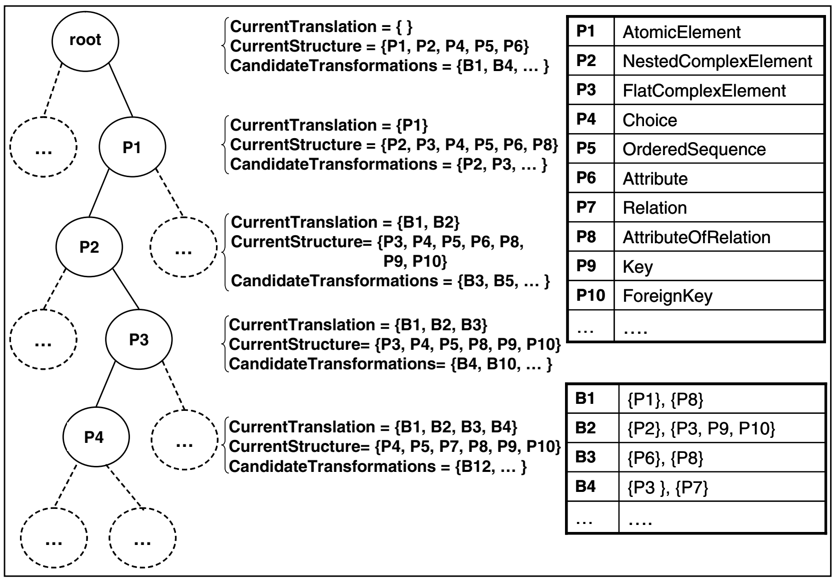
The approach we have followed to limit the search is based on the A^* strategy. This algorithm avoids the expansion of paths of the tree that are already expensive and returns as first result the best solution with respect to an appropriate function $f(n)$ that *estimates* the total cost of a solution. Specifically, the value of $f(n)$ for a node n is obtained as the sum of $g(n)$, the cost of the path so far, and the estimated cost $h(n)$ from n to a valid solution. This function is rather difficult to define in our context. Indeed, the $A*$ search on a tree grants the best solution only if $h(n)$ is admissible, that is, only if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the real cost to reach the solution from n. We have followed here a practical solution: $h(n)$ is defined as a piecewise function that returns zero if the current structure associated with n is empty, and it is proportional to "distance" to the target, that is, the number of patterns of the current structure not occurring in the target model when the set is not empty.

The crucial point is the identification of some heuristics able to limit the search space and generate efficiently good solutions. For each node we consider two properties:

- the size, $l(n)$, that is the length of the current translation in terms of the number of basic transformations that compose it;
- the recall, $r(n)$, which corresponds to the number of patterns of the target model not occurring in the current structure associated with n.

To this end, we have defined a cost function for each node n in the search tree as the linear combination of the two functions: $g(n) = w_1 \times l(n) + w_2 \times r(n)$.

The weights w_1 and w_2 have been chosen such that the recall is privileged to the length of a translation. Intuitively, modifying the w_1 and w_2 we can choose a solution with a better efficiency or a better preferability. Notice that we cannot use the complexity values for the cost function $g(n)$, even if it more precise than the translation size: we must use a value that is comparable to the heuristic $h(n)$, that it is based on the number of patterns, since we cannot know *a priori* the complexity of translation.


root
P1
P2
P3
P4
...
CurrentTranslation = { }
CurrentStructure = {P1, P2, P4, P5, P6}
CandidateTransformations = {B1, B4, ... }
CurrentTranslation = {P1}
CurrentStructure = {P2, P3, P4, P5, P6, P8}
CandidateTransformations = {P2, P3, ... }
CurrentTranslation = {B1, B2}
CurrentStructure= {P3, P4, P5, P6, P8, P9, P10}
CandidateTransformations = {B3, B5, ... }
CurrentTranslation = {B1, B2, B3}
CurrentStructure= {P3, P4, P5, P8, P9, P10}
CandidateTransformations= {B4, B10, ... }
CurrentTranslation = {B1, B2, B3, B4}
CurrentStructure= {P4, P5, P7, P8, P9, P10}
CandidateTransformations = {B12, ... }

P1	AtomicElement
P2	NestedComplexElement
P3	FlatComplexElement
P4	Choice
P5	OrderedSequence
P6	Attribute
P7	Relation
P8	AttributeOfRelation
P9	Key
P10	ForeignKey
...	....

B1	{P1}, {P8}
B2	{P2}, {P3, P9, P10}
B3	{P6}, {P8}
B4	{P3 }, {P7}
...	....

Fig. 4. The search space for the translation

Let us introduce in Figure 4 a practical example to show, in more detail, how the algorithm proceeds. In the example we refer to metaprimitives and patterns with same notation *PX*, see for instance *P2*, that is a ComplexElement that nests another ComplexElement. The example refers to the translation of an XML Schema and its data set into the relational model. Each node of the tree corresponds to a possible state reached by the execution of the algorithm. The first observation is that for each node there could be many candidate transformations and so in principle, to find the optimal solution according to the properties discussed in Section 3, all the possible alternatives should be evaluated. Each node has associated three sets: (i) the set of *PX* (metaprimitives or patterns) involved in the current structure, the *current structure*, (ii) the set of *candidate transformations*, and (iii) the set of transformations collected in the preceding states, that is the *current translation*. As we said, the current structure is initialized to the metaprimitive used in the source schema. For each candidate basic transformation b of a node n, there is a child whose structure is the effect of b

on the current structure of n. A leaf of the tree corresponds to a state in which no candidate transformation exists: if the current structure is not valid for the target model it is a *failing* state. The algorithm stops when the current structure is valid for the target model definition, this would be a *successful* state or a *solution*. In the example at hand, a successful solution is composed by the set of following transformations.

– Atomic elements have been changed into relational attributes.
– Nested elements have been unnested.
– XML attributes have been changed into relational attributes.
– Complex flat elements have been turned into relations with keys.
– Choices have been implemented by means of separate relations.
– Order of sequences have been coded into relational attributes.

The time complexity of the algorithm strictly depends on the heuristic. It is exponential in the worst case, but is polynomial when the heuristic function $h(n)$ meets the condition $|h(n) - h^*(n)| \leq O(\log h^*(n))$, where $h^*(n)$ is the exact cost to get from n to the goal. This condition cannot be guaranteed in our context, since a basic transformation could remove or transform more than just one pattern from the input structure, but in the average case it is polynomial as we show experimentally shortly. More problematic than time complexity can be the memory usage for this algorithm. We have also tried variants of A^* able to cope with this issue, such as memory-bounded $A^*(MA^*)$, but we never hit memory limit even in the original implementation, since the heuristic we used removes effectively not promising paths so that the algorithm does not expand them.

5 Testing the Approach

To evaluate the effectiveness of our approach, we have implemented the translation process within a prototype and we have conducted a series of experiments on several schemas of different data models that vary in terms of dimension and complexity.

In our tests, we have used a specific preference relationship. Consider again the example in Figure 1: the output schema d takes more advantage of the expressiveness of the target model than schemas b or c. It follows that the translation to schema d it is preferable to the other solutions because it also includes the information on cardinalities, which is a construct available in the target data model. For a translation t_x on an input schema S, we denote the number of distinct valid patterns in the target schema by $|t_x(S)|$ and we assume that $t_1 >_{M_t} t_2$ if $|t_1(S)| > |t_2(S)|$.

The results of these experiments are summarized in Figure 5. Each row presents quantitative results for a schema translated from a source model (first column) to a target model (fourth column). The actual number of metaprimitives occurring in the schema is in the second column $|\mathbf{S}|$. The size complexity is estimated by the number of distinct metaprimitives involved in the source schema and it is reported in the third column $|\mathbf{C}|$.

Source Schema	|S|	|C|	Target Model	Num. of sol.	Alg. time	Sol. with min. size	Min. size	Sol. with max. preferabi.	Max. pref.	Number of optimal sol.	A* First Solution	
											Size	Pref
XMLSCHEMA	16	4	ODL	384	2.5	2	3	16	4	18	4	4
XMLSCHEMA	16	4	ER	367	3.4	1	1	4	4	5	2	4
XMLSCHEMA	35	5	DTD	3	2.4	1	1	3	4	1	1	4
XMLSCHEMA	35	5	RELATIONAL	352	2.5	2	3	176	5	2	3	5
RELATIONAL	20	5	XMLSCHEMA	8	1.4	2	3	1	5	1	4	5
RELATIONAL	20	5	ER	2	0.4	1	1	6	5	1	1	5
RELATIONAL	20	5	DTD	6	1.3	2	3	16	5	18	4	5
RELATIONAL	42	6	XMLSCHEMA	24	4.2	1	4	24	6	1	6	6
RELATIONAL	42	6	ODL	8	4.2	1	3	4	6	5	3	5
DTD	27	6	XMLSCHEMA	121	8.5	1	2	89	5	1	2	5
RELATIONAL	54	8	ODL	13	5.1	1	2	4	7	5	2	6
RELATIONAL	54	8	DTD	6	5.4	1	6	6	6	7	7	5

Fig. 5. Experimental results

The first result we report for each translation is the total number of valid possible solutions. It turns out that even for a simple source schema (e.g. a DTD with a few distinct metaprimitives) there are a lot of possible solutions. For instance, valid translations from an XML schema into ODL, as well as into the ER or the relational model, are hundreds. As we said, many of these solutions differ only by the order in which basic transformations occur in the translation, but we still want a criterium to limit the search as discussed in the previous section. Following columns report: (i) the number of solutions with minimal size and the corresponding size value, and (ii) the number of preferable solutions and the corresponding value. We reported the size results instead of the complexity ones to compare them with the A^* results. The optimal solutions are those both efficient and preferable. It turns out that in several cases optimal solutions do not exists and this confirms the intuition that longer solutions may be more "accurate". We also report the overall time (in minutes) the exhaustive search algorithm took to find all the possible solutions for a translation. Notice that this value increases with the number of distinct metaprimitives of the source schema, but in a rather irregular way, since it depends also on the library of transformations and the target model complexity. Indeed, the problem depends on the complexity of the models involved. The required time to find all the solutions becomes critic with models that present more than five distinct metaprimitives. Conversely, the translations involving a restricted number of metaprimitives require a rather limited effort and increase linearly with the complexity of the source schema. Finally, notice in the last two columns the results for the A^* implementation. The first column reports the size of the first solution returned and the second one its preferability. The solution coincides with the optimal for translations between XML based models, whereas in the other cases preferability is close or equal to the best value. We expected this result, since we have tuned the heuristics to maximize the preferability of the solution.

We further evaluated the performance of our best-first search algorithm aggregating results under four main scenarios: (i) translations between database models (Relational, ER, ODL), (ii) between XML based models (XSD, DTD), (iii) from XML based models to database ones and (iv) vice versa. Figure 6 shows the performance of the A^* algorithm on the four scenarios with increasing schema complexity on the x-axis. We ran several examples for each scenario, using schemas with different structures, and reporting average results on the charts. The chart with the size analysis is on the left hand side of the figure. As expected, the number of transformations increases with the number of distinct metaprimitives in the source schema. The linearity in (iii) and (iv) represents the heterogeneity between the two classes of models: for almost each metaprimitive the system uses a transformation from the library. Lines (i) and (ii), instead, show that translations between models that share many metaprimitives require only few basic transformations: this fact strongly depends on the quality of the metamodel implemented in the system. The chart on the right hand side of the figure shows the results of the preferability analysis. The algorithm scales well for the four scenarios: the number of metaprimitives in the target schemas increases with the increasing complexity of the input schemas. In particular, in scenarios (i) and (iv) the system translates with almost linear results: these performances depend on the low complexity of the considered source models, that is, it is easier to find a preferable solution translating a schema from a database model, which presents less constructs and less patterns than an XML data model.

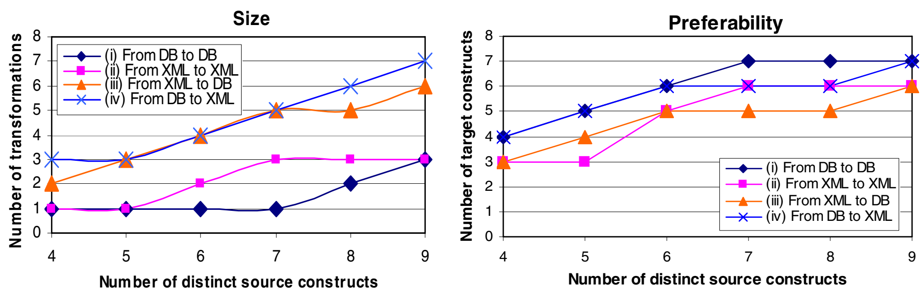


Fig. 6. A^* search strategy analysis

6 Conclusions

In this paper, we have presented new techniques, and supporting results, for the automatic generation of data model translations. We have defined a number of properties for evaluating the quality of the translations and we have experimented them evaluating the translations generated by a prototype. It turned out that the system can generate all the optimal solutions with respect to these properties, but this usually requires significant effort. However, by adopting a best-first search algorithm and appropriate heuristics, a solution can be retrieved efficiently and it is usually optimal in terms of preferability.

In the future, we plan to investigate and develop supporting tools for the design of information preserving transformations. Our work is now focusing on a formal language to express transformation between data models definitions and a graph-based, high-level notation to easily define them.

References

1. Atzeni, P., Cappellari, P., Bernstein, P.A.: Model-independent schema and data translation. In: EDBT. pp.368–385 (2006)
2. Atzeni, P., Torlone, R.: Management of multiple models in an extensible database design tool. In: EDBT, pp. 79–95 (1996)
3. Papotti, P., Torlone, R.: Heterogeneous data translation through xml conversion. J. Web. Eng. 4(3), 189–204 (2005)
4. Fernandez, M.F., Kadiyska, Y., Suciu, D., Morishima, A., Tan, W.C.: Silkroute: A framework for publishing relational data in xml. ACM Trans. Database Syst. 27(4), 438–493 (2002)
5. Bohannon, P., Freire, J., Roy, P., Siméon, J.: From xml schema to relations: A cost-based approach to xml storage. In: ICDE (2002)
6. Ramanath, M., Freire, J., Haritsa, J.R., Roy, P.: Searching for efficient xml-to-relational mappings. In: Xsym, pp. 19–36 (2003)
7. Shu, N.C., Housel, B.C., Taylor, R.W., Ghosh, S.P., Lum, V.Y.: EXPRESS: A Data EXtraction, Processing, amd REStructuring System. ACM TODS 2(2), 134–174 (1977)
8. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. TCS 336(1), 89–124 (2005)
9. Bernstein, P.A.: Applying model management to classical meta data problems. In: CIDR. pp. 209–220 (2003)
10. Bowers, S., Delcambre, L.M.L.: The uni-level description: A uniform framework for representing information in multiple data models. In: ER, pp. 45–58 (2003)
11. Bernstein, P.A., Melnik, S., Mork, P.: Interactive schema translation with instance-level mappings. In: VLDB. pp.1283–1286 (2005)
12. MOF: OMG's MetaObject Facility. `http://www.omg.org/mof/` (2006)
13. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: PODS. pp. 233–246 (2002)
14. Hull, R., King, R.: Semantic database modeling: Survey, applications, and research issues. ACM Comput. Surv. 19(3), 201–260 (1987)
15. Bohannon, P., Fan, W., Flaster, M., Narayan, P.P.S.: Information preserving xml schema embedding. In: VLDB (2005)
16. Hull, R.: Relative information capacity of simple relational database schemata. SIAM J. Comput. 15(3), 856–886 (1986)
17. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: Schema equivalence in heterogeneous systems: bridging theory and practice. Inf. Syst. 19(1), 3–31 (1994)
18. Arenas, M., Libkin, L.: A normal form for xml documents. ACM Trans. Database Syst. 29, 195–232 (2004)
19. Kensche, D., Quix, C., Chatti, M.A., Jarke, M.: GeRoMe: A Generic Role Based Metamodel for Model Management. In: OTM Conferences. pp.1206–1224 (2005)