Toward the Automatic Derivation of XML Transformations

Martin Erwig

Oregon State University
School of EECS
erwig@cs.orst.edu

Abstract. Existing solutions to data and schema integration require user interaction/input to generate a data transformation between two different schemas. These approaches are not appropriate in situations where many data transformations are needed or where data transformations have to be generated frequently.
We describe an approach to an automatic XML-transformation generator that is based on a theory of information-preserving and -approximating XML operations. Our approach builds on a formal semantics for XML operations and their associated DTD transformation and on an axiomatic theory of information preservation and approximation. This combination enables the inference of a sequence of XML transformations by a search algorithm based on the operations' DTD transformations.

1 Introduction

XML is rapidly developing into the standard format for data exchange on the Internet, however, the combination of an ever growing number of XML data resources on the one hand, and a constantly expanding number of XML applications on the other hand, is not without problems. Of particular concern is the danger of isolated data and application "islands" that can lead users to perceive a prodigious supply of data that is often inaccessible to them through their current applications.

This issue has been observed and extensively addressed in previous work in data integration, for example, [8, 14, 6, 7, 19, 13] and more recently in schema integration and query discovery [21, 24, 15, 16]. So far, however, all the proposed solutions require user input to build a translation program or query. Even more troubling, since each different data source requires a separate transformation, the programming effort grows linearly with the number of data sources. In many cases this effort is prohibitive.

Consider the following scenario. An application to evaluate the publication activities of researchers accepts XML input data, but requires the data to be of the form "publications clustered by authors". A user of this system finds a large repository of bibliographic data, which is given in the format according to the DTD shown in Figure 1 on the left. In the following, we will refer to the corresponding XML data as *bib*. The application cannot use these data because

```
<!ELEMENT bib (book|article)*>      <!ELEMENT byAuthor author*>
<!ELEMENT book (title,author*)>     <!ELEMENT author (name,(book|article)*)>
<!ELEMENT article                   <!ELEMENT book title>
       (title,author*,journal)>     <!ELEMENT article (title,journal)>
<!ELEMENT title (#PCDATA)>          <!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA)>         <!ELEMENT title (#PCDATA)>
<!ELEMENT journal (#PCDATA)>        <!ELEMENT journal (#PCDATA)>
```

Fig. 1. DTD of available data and DTD of required data.

bibliographic entries are not grouped by authors. What is needed is a tool that can transform *bib* into a list of author elements, each containing a sublist of their publications. Such a format is shown in Figure 1 on the right.

Although tools are available that support the transformation, they sometimes require non-trivial programming skills. In almost all cases they require some form of user interaction. In any case, users might not be willing to invest their time in generating *one-time* conversion tools. Moreover, if the integration of several different data sources should be required to create several different transformations, the programming or specification effort quickly becomes untenable.

An intrinsic requirement is that these transformations be "as information preserving as possible". In the best case the generated transformation preserves the information content completely, but in many instances transformations that lose information are also sufficient. For example, if an application requires only books with their titles, a transformation that "forgets" the author information of an XML document works well.

Our solution to the described problem can be summarized as follows: First, identify an algebra of information-preserving and information-approximating XML transformations. In particular, these operations have a precisely defined type, that is, an associated schema transformation for DTDs. By induction it then follows that if we transform a DTD d into a DTD d' by a sequence of these elementary XML transformations, the same sequence of operations transforms an XML value of DTD d lossless or approximating into an XML value of DTD d'. The second step is then to define a search algorithm that constructs a search space of DTDs by applying algebra operations and find a path from a source DTD d to the required target DTD d'. The path represents the sequence of operations that realize the sought transformation.

There might be, of course, cases in which the automatic inference does not work well. The situation is comparable to that of search engines like Google that do not always find good matches due to a lack of semantics or structure associated with the query keywords. Nevertheless, search engines are among the most valuable and most frequently used tools of the Internet since they provide satisfactory results in practice. For the same reasons, automatic integration tools, although not complete, might be valuable and useful tools in practice.

This paper presents the proposed approach through examples. Due to space limitations we have to restrict ourselves to the description of a small number of elementary XML operations that can be employed in generated transformations and also a subset of axioms for information approximation. Nevertheless, we will be able to demonstrate the automatic generation of an XML transformation within this restricted setting.

The rest of this paper is structured as follows. In Section 2 we will discuss related work. In Section 3 we will formally define the problem of XML-transformation inference. In Section 4 we axiomatize the notions of information preservation and approximation. In Section 5 we define what it means for an XML transformation to be DTD correct. In Section 6 we introduce basic XML transformations that will be used as building blocks in Section 7 in the inference of complex XML transformations. Finally, Section 8 presents some conclusions.

2 Related Work

Related work has been performed in two areas: (i) schema matching and query discovery and (ii) data semantics and information content.

Schema Matching and Query Discovery. Approaches for matching between different data models and languages are described in [19, 2, 3]. Data integration from an application point of view is also discussed, for example, in [8, 6, 14, 13]. We will not review all the work on data integration here because data integration is traditionally mainly concerned with integrating a set of schemas into a unified representation [22], which poses different challenges than translating between two generally unrelated schemas.

A more specific goal of schema matching is to identify relationships between (elements) of a source and a target schema. Such a mapping can then be used to deduce a transformation query for data.

The Cupid system [15] focuses exclusively on schema matching and does not deal with the related task of creating a corresponding data transformation/query. The described approach combines different methods used in earlier systems, such as MOMIS [4] or DIKE [20].

The Clio system [9] is an interactive, semi-automated tool for computing schema matchings. It was introduced for the relational model in [16] and was based on so-called *value correspondences*, which have to be provided by the user. In [24] the system has been extended by using instances to refine schema matchings. Refinements can be obtained by inferring schema matchings from operations applied to example data, which is done by the user who manipulates the data interactively. User interaction is also needed in [21] where a two-phase approach for schema matching is proposed. The second phase, called *semantic translation*, is centered around generating transformations that preserve given constraints on the schema. However, if few or even no constraints are available, the approach does not work well.

It has been argued in [16] that the computation of schema matchings cannot be fully automated since a syntactic approach is not able to exploit the semantics of different data sources. While this is probably true for arbitrarily complex matches, it is also true that heuristic and linguistic tools for identifying renamings can go a long way [12, 5]. Certainly, quality and sophistication of transformations can be increased by more semantic input. However, there is no research that could quantify the increase/cost ratio. So it is not really known how much improvement is obtained by gathering semantics input. The approach presented in this paper explores the extreme case where users cannot or are not willing to provide input, which means to provide *fully* automatic support for data transformation.

Information Content. A guiding criterion for the discovery of transformations is the preservation (or approximation) of the data sources to which the transformations will be eventually applied. Early research on that subject was performed within relational database theory [10, 11] and was centered around the notion of *information capacity* of database schemas, which roughly means the set of all possible instances that a schema can have. The use of information capacity equivalence as a correctness criterion for schema transformations has been investigated in [17, 18]. In particular, this work provides guidelines as to which variation of the information capacity concept should be applied in different applications of schema translation. One important result that is relevant to our work is that absolute information capacity equivalence is too strong a criterion for the scenario "querying data under views", which is similar in its requirements to data integration. In other words, those findings formally support the use of information approximation in transformation inference.

3 Formalization of Transformation Inference

In the following discussion we make use of the following notational conventions.

Symbols	denote
x, x', y, z	XML elements (also called XML values)
ℓ, ℓ'	lists of XML elements
d, d'	DTDs
t, u	tags
$t_\triangle$	XML elements with tag t
$t[x_1 \ldots x_k]$, $t[\ell]$	XML elements with tag t and subelements $x_1 \ldots x_k$ (or ℓ)

Sometimes we want to refer to a subelement without caring about the exact position of that element. To this end we employ a notation for *XML contexts*: $C\langle x\rangle$ stands for an XML element that contains somewhere a subelement x. Similarly, $C\langle \ell\rangle$ represents an XML element that contains a list ℓ of subelements. This notation is particularly helpful for expressing changes in contexts. To simplify the discussion, we do not consider attributes or mixed content of elements in the following.

Now we can describe the problem of XML-transformation inference precisely as follows. We are given an XML data source x that conforms to a DTD d (which is written as $x : d$), but we need the data in the format described by the DTD d'. Therefore, we are looking for an XML transformation f that, when applied to x, yields an XML value x' that conforms to the DTD d' (that is, $f(x) : d'$) and contains otherwise as much as possible the same information as x. This last condition can be expressed by defining a partial order on XML values $\prec$ that formalizes the notion of having less information content. A slight generalization of the problem is to find transformations f with the described property without knowing x. We can express the problem mathematically as follows.

$$P(d, d') = \{f \mid \forall x. x : d \implies f(x) : d' \wedge \nexists f'. f'(x) : d' \wedge f(x) \prec f'(x)\}$$

P defines the set of all transformations f that map an XML value conforming to d to a value conforming to d' and also have the property that there is no other transformation f' with that property that preserves more information content. The generalized definition reflects the application when the DTD d of the XML data source is known, but the (possibly very large) XML document x has not been loaded (yet). In the following we consider this second case since it subsumes the previous one.

4 Information Preservation and Information Approximation

We formalize the concepts of information preservation and approximation by defining corresponding relations on XML trees. These relations are induced by operations on XML values. We consider here the renamings of tags and regrouping as an information-preserving operation and the deletion of elements as an information-approximating operation. This limitation is not really a problem since the whole theory is generic in the axiomatization of information preservation/approximation, which means that the set of chosen operations does not affect the overall approach.

Formally, two elements that have non-matching tags, such as $x =$ `<t>a</t>` and $x' =$ `<u>a</u>`, are considered to be different. However, if we rename the tag `t` in x to `u`, both elements become identical. We write $\{\mathtt{t} \mapsto \mathtt{u}\}$ for a renaming of `t` to `u` and $\{\mathtt{t} \mapsto \mathtt{u}\}(x)$ for the application of the renaming to the element x. It happens quite frequently that the same data are named differently by different people. For example, we might find bibliographic data sources that wrap the author information by tags `<author>`, `<name>`, `<aname>`, and so on. With regard to the information contained in the XML value, the actual choice of individual tag names does not really matter. Therefore, we can consider a broader kind of equality "up to a tag renaming r", written as $\equiv_r$. For example, under the renaming $\{\mathtt{t} \mapsto \mathtt{u}\}$ the elements x and x' are equal, which we could express, for example, by $x \equiv_{\{\mathtt{t} \mapsto \mathtt{u}\}} x'$. This is because $\{\mathtt{t} \mapsto \mathtt{u}\}(x) = x'$. We must be careful not to rename with a tag that is already in use in the element to be renamed. For example, if we renamed `<author>` to `<title>`, the meaning of the bibliographic data from Section 1 would change. In general, a renaming r can consist of a set of tag renamings, which means that r is a function from old tags to new tags. These two sets can be extracted from a renaming by $\mathit{dom}(r)$ and $\mathit{rng}(r)$, respectively.

We can formalize the equivalence of DTDs modulo renamings by a rule like $\text{REN}_{\equiv}$ shown in Figure 2. In this and the rules to follow, r denotes an arbitrary

$$\text{REN}_{\equiv}\ \frac{rng(r) \cap tags(x) = \varnothing \quad r(x) = x'}{x \equiv_r x'} \qquad \text{CONG}_{\equiv}\ \frac{x_1 \equiv_r y_1 \quad \dots \quad x_k \equiv_r y_k}{t[x_1 \dots x_k] \equiv_r t[y_1 \dots y_k]}$$

$$\text{GRP}_{\equiv}\ \frac{}{C\langle t[\ell_1] \dots t[\ell_k]\rangle \equiv_r C\langle t[\ell_1]\rangle \dots C\langle t[\ell_k]\rangle}$$

$$\text{DEL}_{\preceq}\ \frac{}{C\langle x\rangle \preceq_r C\langle\rangle} \qquad \text{CONG}_{\preceq}\ \frac{x_1 \preceq_r y_1 \quad \dots \quad x_k \preceq_r y_k}{t[x_1 \dots x_k] \preceq_r t[y_1 \dots y_k]}$$

Fig. 2. Axiomatic definition of information content and approximation

(set of) renaming(s). The first premise of the rule prevents name clashes by requiring fresh tags in renamings. The function *tags* computes the set of all tags contained in an XML element. We also have to address the fact that some renamings are more reasonable than others, for example, $\{$`name` $\mapsto$ `aname`$\}$ is more likely to lead to equivalent schemas than, say $\{$`name` $\mapsto$ `price`$\}$. In the described model, any two structurally identical DTDs can be regarded as equivalent under some renaming. This leads to equivalence classes that are generally too large. In other words, schemas that would not be considered equivalent by humans are treated as equivalent by the model. This will be particularly evident when the tags used in the source and target DTD are completely or mostly different.

This problem can be addressed by defining an ordering on renamings that is based on the number and quality of renamings. A cost or penalty can be assigned to each renaming based on its likeliness. For example, names that are "similar" should be assigned a relatively low cost. Measures for similarity can be obtained from simple textual comparisons (for example, one name is the prefix of another), or by consulting a thesaurus or taxonomy like WordNet [1]. Synonyms identified in this way should also have a low penalty. In contrast, any renaming that has no support, such as $\{$`name` $\mapsto$ `price`$\}$, receives a maximum penalty. With this extension we can measure any equivalence $d \equiv_r d'$ by a number, which is given by the sum of the penalties of all renamings in r. Later, we can use this measure to select the "cheapest" among the different possible transformations by favoring a few, well-matching renamings.

Renaming is the simplest form of extending verbatim equality to a form of semantic equivalence. As another example, consider a structural equivalence condition that is obtained from the observation that an element x with tag u containing k repeated subelements with tag t is a grouped or factored representation of the association of each t-element with the rest of x. Therefore, it represents the same information as the corresponding "de-factored" or "ungrouped" representation as k u-elements each containing just one t-element. For instance, the following element on the left represents (in a factored way) the same information as the two elements shown on the right.

```
<book>                               | <book>
  <title>Principia Math.</title>     |   <title>Principia Math.</title>
  <author>Russel</author>            |   <author>Russel</author>
  <author>Whitehead</author>         | </book>
</book>                              | <book>
                                     |   <title>Principia Math.</title>
                                     |   <author>Whitehead</author>
                                     | </book>
```

In general, an element $C\langle t[\ell_1] \ldots t[\ell_k]\rangle$ contains the same information as the list of elements $C\langle t[\ell_1]\rangle \ldots C\langle t[\ell_k]\rangle$. This idea can be captured by the axiom $\textsc{grp}_\equiv$ shown in Figure 2.

Finally, we also need congruence rules to formalize the idea that if elements x and x' contain the same information, then so do, for example, the elements $t[x]$ and $t[x']$. This is achieved by the rule $\textsc{cong}_\equiv$ shown in Figure 2. This approach for formalizing the notion of information equivalence by a set of axioms and rules provides a sound basis for judging the correctness of inferred transformations.

In a similar way, we can axiomatize the notion of *information approximation*. For instance, deleting a subelement from an element x yields a new element x' that contains fewer information than x but agrees otherwise with x. This idea can be expressed by the axiom $\textsc{del}_\preceq$ shown in Figure 2 where we also give a congruence rule $\textsc{cong}_\preceq$ for information approximation. Since the definition of approximation is an extension of equivalence, we also have to account for renamings in the predicate $\preceq_r$.

5 DTD Correctness of XML Transformations

DTDs can be formally defined by extended context-free grammars. Non-recursive DTDs can be represented simply by trees, that is, they can be represented essentially in the same way as XML values. This tree representation simplifies the description of DTD transformations. Note that in this representation `*` and `|` occur as tags. For example, the DTD for *bib* can be represented by the following tree.

$$\texttt{bib}[\texttt{*}[\texttt{|}[\texttt{book}[\texttt{title}, \texttt{*}[\texttt{author}]], \texttt{article}[\texttt{title}, \texttt{*}[\texttt{author}], \texttt{journal}]]]]$$

Representing DTDs as trees means that we can re-use the tree operations we have already defined for XML values. The complexity of the resulting notation can be simplified by abbreviating $\texttt{*}[e]$ by e^* and $\texttt{|}[e, e']$ by $(e|e')$ so that we can recover most of the original DTD notation:

$$\texttt{bib}[(\texttt{book}[\texttt{title}, \texttt{author}^*] \,|\, \texttt{article}[\texttt{title}, \texttt{author}^*, \texttt{journal}])^*]$$

A DTD transformation is given by a function that maps a DTD d to another DTD d'. For each XML transformation f, we can consider its corresponding DTD transformation, for which we write $\overline{f}$. Depending on the language in which

f is defined and on the formalism that is used to describe DTDs and DTD transformations, there might exist zero, one, or more possible DTD transformations for f. The DTD transformation $\overline{f}$ that corresponds to an XML transformation can also be considered as f's *type*, which is expressed by writing $f : d \to d'$ if $\overline{f}(d) = d'$.

Formally relating DTD transformations to the transformations of the underlying XML values is achieved by the notion of *DTD correctness*, that is, an XML operation $f : d \to d'$ is defined to be *DTD correct* if

$$f \text{ applies to } x \quad \Longrightarrow \quad \forall x : d.f(x) : d'$$

In other words, DTD correctness means that the DTD transformation $\overline{f}$ that is associated with an operation f is semantically meaningful, that is, it reflects correctly the DTD transformation for each underlying XML value. (We can write the condition also as: $\forall x : d.f(x) : \overline{f}(d)$.)

6 Basic XML Transformations

The feasibility of the automatic XML-transformation inference hinges to a large part on the ability to express complex XML transformations as compositions of a small set of simple operations, which we call *basic operations*. The design of these basic operations is guided by the following criteria. All basic operations must (a) be information preserving or information approximating, (b) have a clearly specified DTD transformation, and (c) be DTD correct. Why do we require these properties? Item (a) ensures that inferred transformations do not change the information contained in XML data or at most lose information, but never introduce new information. Properties (b) and (c) will ensure that the inference, which is directed by DTDs, yields transformations of XML values that conform to these DTDs. The notion of DTD transformations and correctness will be explained below.

Next we consider three basic XML transformations that have been designed guided the just mentioned criteria: *renaming*, *product*, and *deletion*.

Renaming. The rename operation α takes a renaming $r = \{t_1 \mapsto u_1, \ldots, t_k \mapsto u_k\}$ with $u_i \neq t_i$ for $1 \leq i \leq k$ and applies it to all tags in an XML element x. We require that the new tags u_i do not occur in x.

$$\alpha_r(x) = \begin{cases} r(x) & \text{if } \mathit{rng}(r) \cap \mathit{tags}(x) = \varnothing \\ x & \text{otherwise} \end{cases}$$

Let us check the design constraints for this operation. For information preservation we require that the XML value obtained by the operation in question is equivalent to the original XML value. In the case of renaming we therefore require $\alpha_r(x) \equiv_r x$, which follows directly from the axiom $\text{REN}_{\equiv}$ shown in Figure 2. The DTD transformation that corresponds to renaming can be described by:

$$\alpha_r : d \to r(d)$$

which means that α transforms an XML value conforming to a DTD d into a value whose DTD is obtained by renaming tags according to r. The proof of DTD correctness can be performed by induction over the syntactic structure of the DTD transformation.

Product. Another basic operation is the operation π for de-factoring XML elements. We also call this operation *product* since it essentially computes a combination of an element with a list of its subelements. The tag t of the subelement to be considered is a parameter of π.

$$\pi_t(u[C\langle t[\ell_1]\dots t[\ell_k]\rangle]) = u[C\langle t[\ell_1]\rangle \dots C\langle t[\ell_k]\rangle]$$

The additional root tag u is needed in the definition to force the repetition to apply below the root element. We assume implicitly in this and all other definitions that operations leave all XML values unchanged that do not match the pattern of the definition. In the case of π this means that for any element x that does not contain repeated t-subelements we have $\pi_t(x) = x$. Again we can check the properties of the operation π. First, information preservation follows from the axiom $\textsc{grp}_\equiv$ and the congruence rule $\textsc{cong}_\equiv$ shown in Figure 2. The type of π is:

$$\pi_t : u[C\langle t^*_\triangle\rangle] \to u[C\langle t_\triangle\rangle^*]$$

DTD correctness can again be shown by induction.

Deletion. As an example for an information-approximating operation, consider the XML transformation δ_t that deletes a sequence of t-subelements (on one level) from an XML element. It can be defined as follows.

$$\delta_t(C\langle t[\ell_1]\dots t[\ell_k]\rangle) = C\langle\rangle$$

Obviously, δ is *not* information preserving, but it is information approximating, which can be proved using the axiom $\textsc{del}_\preceq$ from Figure 2. The type of δ can be described succinctly by re-using the context notation for XML trees.

$$\delta_t : C\langle t^*_\triangle | t_\triangle\rangle \to C\langle\rangle$$

As for the other XML transformations, DTD correctness can be proved by induction.

To summarize, for all the basic operation ω defined, we have the following property.

$$\forall x.x : d \implies \omega(x) : \overline{\omega}(d) \wedge (\exists r.x \equiv_r \omega(x) \vee \omega(x) \preceq_r x)$$

That is, each basic operation ω is: (1) DTD correct and (2a) information preserving or (2b) information approximating (recall that $\overline{\omega}$ denotes the DTD transformation of ω).

7 Transformation Inference

A very simple, although effective, initial approach is to build a search space of DTDs starting from the DTD of the source document, say d, by repeatedly applying all matching operations until the target DTD, say d', is reached. By "matching operations" we mean basic operations whose argument type have d as an instance.

In the search we always favor following paths along information-preserving operations over information-approximating operations. Whenever we apply α we take $tags(d')$ as a pool from which to draw new names. We also have to ensure not to repeatedly apply inverse renamings to prevent running into infinite search paths.

Once we have reached d' by this procedure, the path from d to d' in this search space corresponds to a sequence of basic XML transformations $\omega_1, \ldots, \omega_k$ whose composition $f = \omega_k \cdot \ldots \cdot \omega_1$ is the sought transformation of type $d \to d'$. This is because we are using only DTD-correct transformations. If all basic operations ω_i are information preserving, then so is the transformation f. If at least one ω_i is information approximating, then so is f. If we are not able to generate d', the algorithm stops with an error.

To illustrate the transformation inference by an example, consider the task of creating a list of title/author pairs for books from the *bib* element. This means to find a transformation from the DTD d for *bib*

$$\texttt{bib}[(\texttt{book}[\texttt{title}, \texttt{author}^*] \mid \texttt{article}[\texttt{title}, \texttt{author}^*, \texttt{journal}])^*]$$

into the following DTD d'.

$$\texttt{bookAuthors}[\texttt{book}[\texttt{title}, \texttt{author}]^*]$$

First, since the tag `bookAuthors` is not contained in the source DTD d, we know that we have to apply α_r with $r = \{\texttt{bib} \mapsto \texttt{bookAuthors}\}$. Next, we can apply $\delta_{\texttt{article}}$ because its type matches with the context

$$C_1 = \texttt{bookAuthors}[(\texttt{book}[\texttt{title}, \texttt{author}^*] \mid \langle\,\rangle)^*]$$

However, we might also apply $\pi_{\texttt{author}}$ by choosing, for example, the following context (note that $u = \texttt{bookAuthors}$).

$$C_2 = (\texttt{book}[\texttt{title}, \texttt{author}^*] \mid \texttt{article}[\texttt{title}, \langle\,\rangle, \texttt{journal}])^*$$

(Alternatively, we could also match $\texttt{author}^*$ in the `book` element.) Nevertheless, we choose to apply δ because it is simpler, which is somehow indicated by the smaller context C_1. We could also try to apply $\delta_{\texttt{book}}$ to delete the `book` element, which, however, does not seem to make any sense because we then "lose" a tag of the target DTD. After having applied $\delta_{\texttt{article}}$, we have reached the DTD described by the context C_1. Now it makes sense to apply $\pi_{\texttt{author}}$. Before we do this, however, we simplify C_1 according to a rule $d|\langle\,\rangle = d$ to remove the now unnecessary $|$ constructor. So the context for the application of $\pi_{\texttt{author}}$ is (with $u = \texttt{bookAuthors}$):

$$C_3 = \texttt{book}[\texttt{title}, \langle\,\rangle^*]^*$$

The resulting DTD after the application of $\pi_{\texttt{author}}$ is

$$\texttt{bookAuthors}[(\texttt{book}[\texttt{title}, \texttt{author}]^*)^*]$$

A final simplification through the rule $(d^*)^* = d^*$ [23] yields the target DTD. The inference process has therefore generated the transformation

$$f = \pi_{\texttt{author}} \cdot \delta_{\texttt{article}} \cdot \alpha_{\{\texttt{bib} \mapsto \texttt{bookAuthors}\}}$$

The description is a bit simplified, because in order to apply the operations in f to some XML value, we need all the contexts that were determined during the inference process. Treating these contexts here like implicit parameters, we can now apply f to *bib* and obtain the desired XML value.

With two additional operations for lifting elements upward in XML trees and grouping elements according to common subelements, we can describe the XML transformation that is required for the example given in Section 1. Designing these operations so that they are DTD correct and information preserving/approximating and making transformation inference powerful enough to discover them is part of future work.

8 Conclusions

The fast growing number of Web applications and available information sources carries the danger of creating isolated data and application islands because the distributed nature of the Internet does not enforce the use of common schemas or data dictionaries. Our approach aims at avoiding these data islands and to promote the free flow and integration of differently structured data by developing a system for the automatic generation of XML transformations.

Our approach differs from previous efforts since we aim at a fully automated transformation discovery tool where user interaction is not required a priori. It will not, however, rule out any additional input the user is willing to provide. As one example, user-defined renamings can be easily integrated into our approach by setting penalties for these renamings to zero. In other words, users can interact if they want to, but are not required to do so.

References

1. WordNet: A Lexical Database for the English Language. `http://www.cogsci.princeton.edu/~wn/`.
2. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and Translation for Heterogeneous Data. In *6th Int. Conf. on Database Theory*, LNCS 1186, pages 351–363, 1997.
3. P. Atzeni and R. Torlone. Schema Translation between Heterogeneous Data Models in a Lattice Framework. In *6h IFIP TC-2 Working Conf. on Data Semantics*, pages 345–364, 1995.
4. S. Bergamaschi, S. Castano, and M. Vincini. Semantic Integration of Semistructured and Structured Data Sources. *SIGMOD Record*, 28(1):54–59, 1999.
5. M. W. Bright, A. R. Hurson, and S. Pakzad. Automated Resolution of Semantic Heterogeneity in Multidatabases. *ACM Transactions on Database Systems*, 19(2):212–253, 1994.

6. V. Christophides, S. Cluet, and J. Simèon. On Wrapping Query Languages and Efficient XML Integration. In *ACM SIGMOD Conf. on Management of Data*, pages 141–152, 2000.
7. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators Need Data Conversion! In *ACM SIGMOD Conf. on Management of Data*, pages 177–188, 1998.
8. A. Eyal and T. Milo. Integrating and Customizing Heterogeneous E-Commerce Applications. *VLDB Journal*, 10(1):16–38, 2001.
9. L. M. Haas, R. J. Miller, B. Niswonger, M. T. Roth, P. M. Schwarz, and E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 22(1):31–36, 1999.
10. R. Hull. Relative Information Capacity of Simple Relational Database Schemata. *SIAM Journal of Computing*, 15(3):856–886, 1986.
11. T. Imielinski and N. Spyratos. On Lossless Transformation of Database Schemes not Necessarily Satisfying Universal Instance Assumption. In *3rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 258–265, 1984.
12. P. Johannesson. Linguistic support for Analysing and Comparing Conceptual Schemas. *IEEE Transactions on Knowledge and Data Engineering*, 21(2):165–182, 1997.
13. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *22nd Int. Conf. on Very Large Databases*, pages 251–262, 1996.
14. B. Ludäscher, Y. Papakonstantinou, and P. Velikhov. Navigation-Driven Evaluation of Virtual Mediated Views. In *7th Int. Conf. on Extending Database TechnologyEuropean*, LNCS 1777, pages 150–165, 2000.
15. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *27th Int. Conf. on Very Large Databases*, pages 49–58, 2001.
16. R. J. Miller, L. M. Haas, and M. A. Hernàndez. Schema Mapping as Query Discovery. In *26th Int. Conf. on Very Large Databases*, pages 77–88, 2000.
17. R. J. Miller, Y. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *19th Int. Conf. on Very Large Databases*, pages 120–133, 1993.
18. R. J. Miller, Y. Ioannidis, and R. Ramakrishnan. Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice. *Information Systems*, 19(1):3–31, 1994.
19. T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *24th Int. Conf. on Very Large Databases*, pages 122–133, 1998.
20. L. Palopoli, G. Terracina, and D. Ursino. Towards the Semi-Automatic Synthesis of Cooperative Information Systems and Data Warehouses. In *ADBIS-DASFAA Symp. on Advances in Databases and Information Systems*, pages 108–117, 2000.
21. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernàndez, and R. Fagin. Translating Web Data. In *28th Int. Conf. on Very Large Databases*, 2002.
22. S. Ram and V. Ramesh. Schema Integration: Past, Current and Future. In A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, editors, *Management of Heterogeneous and Autonomous Database Systems*, pages 119–155. Morgan Kaufman, 1999.
23. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *25th Int. Conf. on Very Large Databases*, pages 302–314, 1999.
24. L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. In *ACM SIGMOD Conf. on Management of Data*, 2001.