A ToolBox for Conservative XML Schema Evolution and Document Adaptation

Joshua Amavi, Jacques Chabin, Mirian Halfeld Ferrari, and Pierre Réty

Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 Orléans, France
{joshua.amavi,jacques.chabin,mirian,pierre.rety}@univ-orleans.fr

Abstract. We propose an algorithm that computes a mapping to obtain a conservative extension of original local schemas. This mapping ensures schema evolution and guides the construction of a document translator.

1 Introduction

Our goal is to establish a multi-system environment composed by a global central system which is a *conservative* evolution of local ones, capable of processing changes that can then be transmitted to local systems. The communication should be possible in both directions: local-to-global and global-to-local. We allow independent local services to continue working on their own data, with their own tools while permitting diagnosis and changes based on a general and complete view of all services. This scenario requires tools for dealing with type evolution and document adaptation. It can be useful as a temporary configuration, deferring complete integration until local systems are ready, or as a flexible architecture adopted by the enterprise. In this context, we suppose that $S_1, \ldots, S_n$ are local systems which deal with sets of XML documents $X_1, \ldots, X_n$, respectively, and that inter-operate with a global, integrated system S. Each set X_i conforms to schema or type constraints $\mathcal{D}_i$, while $\mathcal{D}$ is an extended type (of S) that accepts any local document from $\mathcal{D}_i$. We assume that the global system S may evolve to S', accepting more documents or rejecting some original ones. Our goal is to propose tools allowing automatic type transformation accompanied by automatic document translation. We implement a platform[1] where all our proposed tools will be available (refer to [3] for details):

- *ExtSchemaGenerator*([5]) extends a given schema G, seen as a regular tree grammar, into a new grammar G' respecting the following property: the language generated by G' is the smallest set of unranked trees that contains the language generated by G and the grammar G' is a Local Tree Grammar (LTG) or a Single-Type Tree Grammar (STTG).

[1] From previous work: *ExtSchemaGenerator*, available on `http://www.univ-orleans.fr/lifo/Members/rety/logiciels/RTGalgorithms.html` *XMLCorrector*, available on `http://www.info.univ-tours.fr/~savary/English/xmlcorrector.html`

H. Decker et al. (Eds.): DEXA 2014, Part I, LNCS 8644, pp. 299–307, 2014.

- *XMLCorrector*([2]) corrects an XML document *w.r.t.* schema constraints expressed as a DTD (or an LTG). The corrector reads the entire XML tree t to propose solutions. *XMLCorrector* finds all solutions within a given threshold th.
- *MappingGen*: We propose an algorithm that applies the ideas of [5] to generate a mapping from one schema G, seen as a regular tree grammar, to an extended schema G' which will be an LTG. The resulting schema mapping m is a sequence of operations on grammar rules that indicates, step by step, how to transform G into G' following the approach in [5]. Given a mapping m we can easily compute its inverse m^{-1} or compose it to other mappings; allowing schemas to evolve.
- *XTraM*: Based on a given mapping m (from schema S to T), we propose a method to translate an XML document (or tree) t, valid *w.r.t.* S into a document t' valid *w.r.t.* T. The edit distance between t and t' is no higher than a given positive threshold th. Moreover, t' is the closest tree to t, obtained by changing t according to the schema modifications imposed by m. For each edit operation on S, to obtain T, we analyse what should be the corresponding update on document t. When this update violates validity, we use *XMLCorrector* to propose corrections to the subtree involved in the update.

2 Schema Evolution

An XML document is an *unranked tree*, defined in the usual way as a mapping t from a set of positions $Pos(t)$ to an alphabet Σ. We deal with XML schema represented as RTG (regular tree grammar) $G = (N, \Sigma, S, P)$, where: N is a finite set of *non-terminals*; Σ is a finite set of *terminals*; S is a set of *start symbols*, where $S \subseteq N$ and P is a finite set of *production rules* of the form $X \to a\,[R]$, where $X \in N$, $a \in \Sigma$, and R is a regular expression over N. As usual, in this paper, our algorithms start from grammars in reduced and normal form. Among RTG we are interested in local tree grammars (LTG) which have the same expressive power as DTD. We refer to [5] for details.

Conservative XML Type Extension (ExtSchemaGenerator). In order to compute an LTG that extends minimally a given RTG, we follow the idea of *ExtSchemaGenerator*, presented in [5]. This method is very simple when dealing with the generation of an LTG from an RTG: replace each pair of competing non-terminals by a new non-terminal, until there are no more competing non-terminals. The regular expression of a new non-terminal rule is the disjunction of the regular expressions associated to competing non-terminals.

Consider three hospital services (patient and treatment service, insurance service and bill service), each one having its own LTG (or DTD) as schema. Figure 1(lines 1-5) shows the RTG obtained by the union of the production rules of all these three grammars while Figure 1(lines 3-6) shows the resulting LTG. The obtained LTG is an extension of the original RTG since it generates *all* trees generated by the original RTG and possibly others as well (refer to example of Figure 3). Clearly, the obtained grammar is also an extension of each hospital

1	$H_1 \rightarrow hospital[I_1^*]$	$H_2 \rightarrow hospital[I_2^*]$	$H_3 \rightarrow hospital[I_3^*]$
2	$I_1 \rightarrow info[P \mid T]$	$I_2 \rightarrow info[C \mid Pol]$	$I_3 \rightarrow info[B]$
3	$P \rightarrow patient[S \cdot N \cdot V^*]$	$C \rightarrow cover[S \cdot PN]$	$B \rightarrow bill[S \cdot It^* \cdot D]$
4	$V \rightarrow visitInfo[Id \cdot D]$	$Pol \rightarrow policy[PN \cdot Id^*]$	$It \rightarrow item[Id \cdot PZ]$
5	$T \rightarrow treatment[Id \cdot TN \cdot PR]$	$PR \rightarrow procedure[T^*]$	
6	$H_1 \rightarrow hospital[I_1^* \mid I_1^* \mid I_1^*]$	$I_1 \rightarrow info[(P \mid T) \mid (C \mid Pol) \mid B]$	

Fig. 1. Lines 1-5 : RTG obtained from the union of production rules of grammars. Lines 3-6 : LTG obtained by algorithm in [5] from the RTG (lines 1-5).

service grammar. In Figure 3(a) we find a document valid *w.r.t.* the billing local schema. It is also valid *w.r.t.* the global schema of Figure 1(lines 3-6).

Schema Mappings. We say that a *source* schema (or grammar) evolves to a *target* schema. A *schema mapping* is specified by an operation list, denoted as an *edit script*, that should be performed on source schema in order to obtain the target schema. We propose an algorithm that generates a mapping to translate an RTG G into an LTG G', following the lines of [5] . Our mapping is composed by a sequence of *edit operations* that should be applied on the rules of grammar G in order to obtain G'.

In the following definition, let *ed* be an *edit operation* defined on RTG G. We denote by $ed(G)$ the RTG obtained by applying *ed* on G. Each edit operation is associated with a cost that can be fixed according to the user's priority. The cost of an edit script is the sum of the costs of the edit operations composing it.

Definition 1 (Edit Script). An *edit script* $m = \langle ed_1, ed_2, \ldots ed_n \rangle$ is a sequence of *edit operations* ed_k where $1 \leq k \leq n$. Let G be an RTG, an *edit script* $m = \langle ed_1, ed_2, \ldots ed_n \rangle$ is defined on G iff there exists a sequence of RTG $G_0, G_1, \ldots, G_n$ such that: (i) $G_0 = G$ and (ii) $\forall\, 1 \leq k \leq n$, ed_k is defined on G_{k-1} and $ed_k(G_{k-1}) = G_k$. Hence, $m(G) = G_n$. The empty *edit script* is denoted $\langle\rangle$. The cost of an edit script m is defined as $cost(m) = \Sigma_{i=1}^{n}(cost(ed_i))$. □

Definition 2 (Schema Mapping). A schema mapping is a triple $\mathcal{M} = (S, T, m)$, where S is the source schema, T is the target schema, and m is an *edit script* that transforms S into T (*i.e.*, $m(S) = T$). We say that $\mathcal{M}$ is syntactically *specified by*, or, *expressed* by m. □

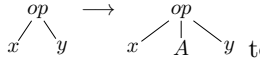
Edit Operations. In this section, the problem of changing one RTG into another is treated as a tree editing problem.

Tree Representation for Production Rules. We represent the right-hand side of a production rule $X \rightarrow a\,[R]$ as a tree denoted t_X^r such that $t_X^r = a(t_R)$. The root of t_X^r is the terminal a which has only one subtree t_R. For example, in Figure 2, the tree on the top left corner is t_I^r with $I \rightarrow info[T.(Y.Co)]$.

Definition 3 (Well Formed Tree). A tree t representing the right-hand side of a production rule is well formed iff the following conditions are verified: (i) the root is a terminal symbol with only one child; (ii) the leaves nodes are in

$N \cup \{\epsilon\}$ and (*iii*) the internal nodes are in the set $\{|, ., *\}$ having exactly one child (if it is in $\{*\}$) or at least one child, in the other cases. □

Elementary Edit Operations. In [3], we define elementary edit operations by using rewriting. Here we give an intuition of some of these operations. Given an RTG $G = (N, \Sigma, S, P)$ in normal form, an *elementary edit operation ed* is a partial function that transforms G into a new RTG G'. The *elementary edit operation ed* can be applied on G only if *ed* is defined on G. Below we distinguish four types of *elementary edit operations* on RTG, where $A \in N$:

1. Edit operations to modify the set of start symbols S: `set_startelm`(A) and `unset_startelm`(A) to add/delete the non-terminal A to/from S.
2. Edit operations to modify non-terminal or terminal symbols in a content model. For example, `ins_elm`$(X, A, u.i)$: (cf. Figure 2(ed_1)) applies the rewrite rule  to insert A in the position $u.i$ of tree representing the production rule associate to X;
   `del_elm`$(X, A, u.i)$: (cf. Figure 2(ed_2)) is the inverse operation. The other operations are: `rel_root`(X, a, b): (cf. Figure 2(ed_3)) to rename the root from a to b; `rel_elm`(X, A, B, u): (cf. Figure 2(ed_4)) to rename non-terminal A at position u into B.
3. Edit operations to modify operator symbols in a content model are similar to those described in the previous item, but apply on operator nodes. See for instance Figure 2(ed_5), (ed_6) and (ed_7).
4. Edit operations to modify the set of production rules P are: `ins_rule`(A, a) that adds the new production rule $A \to a\,[\epsilon]$ to P and the non-terminal A to S, where $A \notin N$ and `del_rule`(A, a) its inverse.

After each edit operation, the sets Σ and N are automatically updated to contain all and only the terminal (resp. non-terminal) symbols appearing in P.

Proposition 1. *Let G and G' be two RTG. There exist an edit script, composed only by elementary edit operation, that transforms G into G'.* □

Non-elementary Edit Operations. For readability and cost estimation, we define short-cut operations, *i.e.*, operations seen as a one-block operation but equivalent to a sequence of *elementary edit operations*, such as `ins_tree`$(X, R, u.i)$, `ins_treerule`(A, a, R) and inversely `del_tree`$(X, R, u.i)$, `del_treerule`(A, a, R).

Operation Cost. For each *edit operation ed*, we define a non-negative and application-dependent cost. On the one hand, we assume that operations that do not change the language generated by the RTG G on which they were applied, are 0-cost. Their goal is just to simplify a given regular expression. For instance, `del_opr`$(X, opr, u.i)$ where $t^r_X(u) = t^r_X(u.i) = opr$ and `del_opr`$(X, opr, u.i)$ where $t^r_X(u.i) \in \{|, .\}$ and $t^r_X(u.i)$ has exactly one child, are 0-cost operations. On the other hand, we suppose that an *elementary edit operation* costs 1, while a *non-elementary edit operation* costs 5.

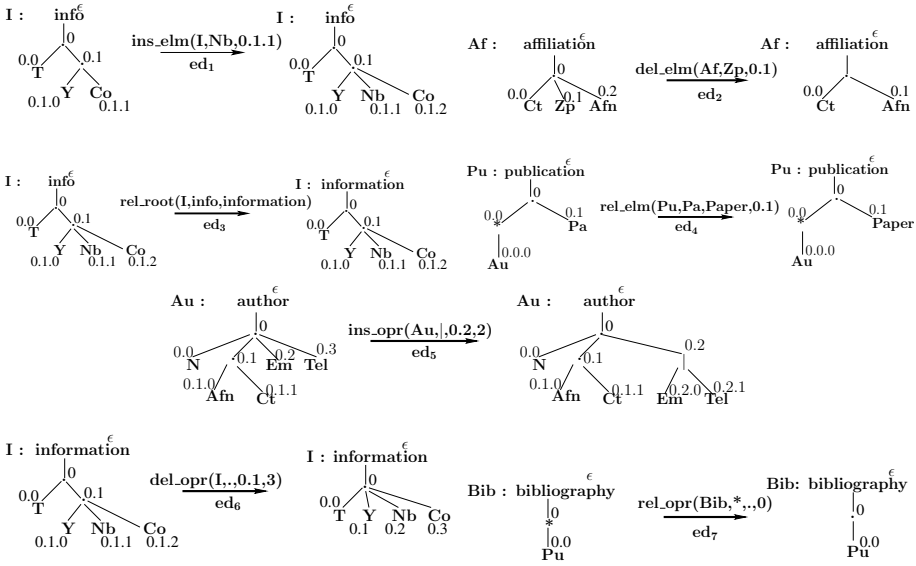


Fig. 2. Example of *elementary edit operations*

Generating a Schema Mapping (MappingGen). Algorithm 1 generates a mapping that converts an RTG in an LTG by following the ideas in [5], explained in Section 2. This algorithm starts by determining a set of competing non-terminals EC_a (lines 2-3). Then we can take arbitrarily in EC_a, one of these non-terminals (say X_0) to represent all others, *i.e.*, when merging rules of competing terminals, one non-terminal name is chosen to represent the result of the merge (line 4). Recall that edit operations always deal with a production rule in its tree-like format. The new production rule of X_0 is built in two steps. We add an OR operation as the parent of its original regular expression $reg(X_0)$ (line 5) and then we insert all regular expressions associated with its competing non-terminals as siblings of $reg(X_0)$ (line 7). In line 8 we just replace, in all production rules, non-terminals in EC_a by X_0. Original rules of non-terminals in EC_a are deleted (line 10) after, possibly, adjusting start symbols (line 9).

Proposition 2. *Let m be the mapping obtained by Algorithm 1 from an RTG G. The language $L(m(G))$ is the least LTL that contains $L(G)$. Moreover, the grammar $m(G)$ equals the one obtained by ExtSchemaGenerator.* □

Going Further with Mappings to Support Schema Evolution. In [6], it was shown how two fundamental operators on schema mappings, namely composition and inversion, can be used to address the mapping adaptation problem in the context of schema evolution. Let $\mathcal{M}_1$ be a mapping between XML schemas S and T. When S or T evolve, $\mathcal{M}_1$ shall be adapted. By using composition and inversion operators, one can avoid mapping re-computation. We now precise the notions of composition and inversion in our context.

Algorithm 1. A mapping for transforming an RTG into an LTG

Input: A Regular Tree Grammar $G = (NT, \Sigma, S, P)$
Output: An *edit script* m between G and the LTG G' such that $L(G) \subseteq L(G')$

```
1: m := ⟨⟩
2: for each terminal symbol a ∈ Σ do
3:     EC_a = {X_0, ..., X_k} is a set of competing non-terminals where term(X_i) = a
4:     Non-terminal X_0 is choosed to represent X_0, ..., X_k
5:     Add ins_opr(X_0, |, 0, 1) to m
6:     for each non-terminal X_i ∈ {X_1, ..., X_k} do
7:         Add ins_tree(X_0, reg(X_i), 0.i) to m
8:         Add rel_elm(Y, X_i, X_0, u) to m, for all u where u is the position of X_i
                                        in the rule Y → b [R] ∈ P
9:         Add set_startelm(X_0) to m where X_0 ∉ S and X_i ∈ S
10:        Add del_treerule(X_i, a, reg(X_i)) to m
11:    end for
12: end for
13: return m
```

Definition 4 (Mapping composition and inversion). Given two mappings $\mathcal{M}_1 = (S, T, m_1)$ and $\mathcal{M}_2 = (T, V, m_2)$, the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is the mapping $\mathcal{M}_1 \circ \mathcal{M}_2 = (S, V, m_1 \,.\, m_2)$. If $m_1 = \langle ed_1, \cdots, ed_n \rangle$, then the inverse of mapping $\mathcal{M}_1$ is the mapping $\mathcal{M}_1^{-1} = (T, S, m_1^{-1})$ where $m_1^{-1} = \langle ed_n^{-1}, \cdots, ed_1^{-1} \rangle$ and $ed_k^{-1} (1 \leq k \leq n)$ is defined in [3][2].

3 Adapting XML Documents to a New Type

Correcting XML Documents (XMLCorrector) In [2], given a well-formed XML tree t, a schema G and a non negative threshold th, *XMLCorrector* finds every tree t' valid *w.r.t.* G such that the edit distance between t and t' is no higher than th. Contrary to most other approaches, [2] considers the correction as an enumeration problem rather than a decision problem and computes all the possible corrections on t. The algorithm, proved to be correct and complete in [2], consists in fulfilling an edit distance matrix which stores the relevant edit operation sequences allowing to obtain the corrected trees. The theoretical exponential complexity of *XMLCorrector* is related to the fact that edit sequences and the corresponding corrections are generated and that the correction set is complete.

In this paper, contrary to [2], we do not consider all the possible corrections on t. The correction of XML documents is guided by a given mapping. For each edit operation on S, to obtain T, we analyse what should be the corresponding update on document t. When this update violates validity, we use *XMLCorrector* to propose corrections to the subtree involved in the update.

[2] Notice that inverses are defined in the intuitive way as, for example, $\texttt{ins\_elm}(X, A, u) \rightleftharpoons \texttt{del\_elm}(X, A, u)$ or $\texttt{rel\_opr}(X, p, q, u) \rightleftharpoons \texttt{rel\_opr}(X, q, p, u)$.

Document Translation Guided by Mapping (XTraM). Our method consists in performing a list of changes on XML documents, in accordance with the edit operations found in the mapping. For example, adding or deleting a regular expression in a rule under the operator '.' is a mapping operation that provokes, respectively, the insertion or the deletion of a subtree in an originally valid XML tree (to maintain its validity). Similarly, renaming a non-terminal A by B, provokes the substitution of the subtree generated by A into the subtree generated by B. When local correction on XML subtrees are needed, *XMLCorrector* is used to ensure document validity.

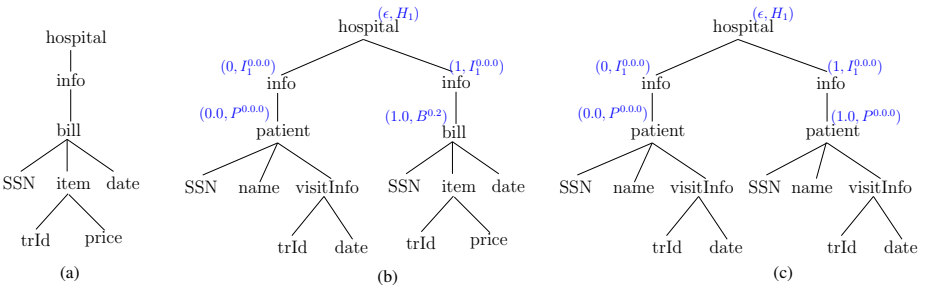


Fig. 3. (a) XML tree valid *w.r.t.* the billing local schema. (b) XML tree valid *w.r.t.* the global schema of Figure 1(lines 3-6). (c) Tree resulting from the translation of (b) into the patient local schema (cf. Figure 1(lines 1-5)). Trees (b) and (c) are annotated.

Consider an XML tree t valid *w.r.t.* schema S and a mapping m from S to T. Our method can be summarized in two steps:

1. Since t belongs to the language $L(S)$, it is possible to associate a non-terminal A with each tree node position p generated by this non-terminal. We analyse t, detect each non-terminal and annotate it with its corresponding position u in the used production rule. This annotation respects the format (p, A^u). For example, in Figure 3(b), we notice that the tree node *bill* is generated by the non-terminal B whose position in $t^r_{I_1}$ is 0.2, noted as $(1.0, B^{0.2})$.
2. Each edit operation ed in m activates a set of modifications on t. When ed transforms a grammar into a new grammar containing the previous one, the set of modifications is empty. Otherwise, our method consists in traversing t (marked as in step 1) in order to find the tree positions which may be affected due to ed. Modifications on t are defined according to each edit operation and are not detailed here due to the lack of space. Obviously, if no position is affected, t does not change.

Figure 3(b) shows an XML document concerning patients and bills. This document is valid *w.r.t.* the global schema but not valid *w.r.t.* to any local schemas. Translating the document of Figure 3(b) into a document respecting the patient schema we obtain the document of Figure 3(c).

4 Related Work and Concluding Remarks

Much other work deals with schema evolution. In [6] second order logic is needed to express some mapping compositions. This approach is the basis of [1,8,12]. We believe that the use of edit operations makes our approach simpler than theirs and gets on well with our previous work concerning XML document correction. Proposals, such as those in [7,9,10,4,11], use edit operations. ELaX [10] and Exup [4] are a domain-specific language that proposes to handle modifications on XSD and to express such modifications formally. An important originality of our approach is the *automatic* generation of a *conservative extension* of an RTG into an LTG and the fact that it may propose different solutions to be chosen by the user. *XTraM* is guided by a mapping and produces documents with corrections that do not exceed a threshold.

Our *ToolBox* offers schema evolution mechanisms accompanied by an automatic adaptation of XML documents. Its conservative aspect guarantees great flexibility when a global integrated system co-exists with local ones. A prototype, implemented in Java, is been tested. As a first experiment, we have produced an LTG, in $24ms$, by merging the grammars obtained from *dblp* DTD[3] and *HAL XSD*[4]. *MappingGen* returned a 19-operation mapping. Then *XTraM* was used to adapt a 52-node document valid *w.r.t.* the computed LTG toward the *HAL* grammar, giving, in this case, 36 solutions in 22.6 s. As in this test, all possible translations can be considered, but the user may also interfere in an intermediate step, making choices before the end of the complete computation - guiding and, thus, restricting the number of solutions.

References

1. Amano, S., Libkin, L., Murlak, F.: XML schema mappings. In: Proceedings of the 28th ACM Symposium on Principles of Database Systems, PODS 2009, pp. 33–42. ACM, New York (2009)
2. Amavi, J., Bouchou, B., Savary, A.: On correcting XML documents with respect to a schema. The Computer Journal 56(4) (2013)
3. Amavi, J., Chabin, J., Halfeld Ferrari, M., Réty, P.: A toolbox for conservative XML schema evolution and document adaptation. CoRR abs/1406.1423 (2014)
4. Cavalieri, F., Guerrini, G., Mesiti, M.: Updating XML schemas and associated documents through Exup. In: Proceedings of the IEEE 27th International Conference on Data Engineering, ICDE 2011, pp. 1320–1323. IEEE Computer Society, Washington, DC (2011)
5. Chabin, J., Halfeld Ferrari, M., Musicante, M.A., Réty, P.: Conservative Type Extensions for XML Data. In: Hameurlain, A., Küng, J., Wagner, R. (eds.) TLDKS IX. LNCS, vol. 7980, pp. 65–94. Springer, Heidelberg (2013)
6. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Schema mapping evolution through composition and inversion. In: Bellahsene, Z., Bonifati, A., Rahm, E. (eds.) Schema Matching and Mapping, pp. 191–222. Springer (2011)

[3] `http://dblp.uni-trier.de/xml/dblp.dtd`
[4] `http://import.ccsd.cnrs.fr/xsd/generationAuto.php?instance=hal`

7. Horie, K., Suzuki, N.: Extracting differences between regular tree grammars. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC 2013, pp. 859–864. ACM, New York (2013)
8. Jiang, H., Ho, H., Popa, L., Han, W.S.: Mapping-driven XML transformation. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 1063–1072. ACM, New York (2007)
9. Leonardi, E., Hoai, T.T., Bhowmick, S.S., Madria, S.K.: Dtd-diff: A change detection algorithm for dtds. Data Knowledge Engineering 61(2), 384–402 (2007)
10. Nösinger, T., Klettke, M., Heuer, A.: XML schema transformations. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013, Part I. LNCS, vol. 8055, pp. 293–302. Springer, Heidelberg (2013)
11. Suzuki, N., Fukushima, Y.: An XML document transformation algorithm inferred from an edit script between DTDs. In: Proceedings of the 19th Conference on Australasian Database, ADC 2008, vol. 75, pp. 175–184. Australian Computer Society, Inc., Darlinghurst (2007)
12. Yu, C., Popa, L.: Semantic adaptation of schema mappings when schemas evolve. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005, pp. 1006–1017. VLDB Endowment (2005)