Assisting XML Schema Evolution that Preserves Validity

Béatrice Bouchou[1], **Denio Duarte**[2]

[1]Université François Rabelais de Tours - LI/Campus Blois - France

[2]Universidade Comunitária Regional de Chapecó / Unochapecó - CETEC (SC) Brazil

`beatrice.bouchou@univ-tours.fr, denio@unochapeco.edu.br`

Abstract. *We consider the problem of XML schema evolution preserving the validity of existing documents related to the original schema. The aim of such schema evolution is to fit new needs without revalidating all existing valid XML documents. We propose an approach to assist users to specify schema updates that have no impact on existing document validity. An XML schema is modeled as a set of regular expressions, each constraining the content model of XML elements. Given the user needs, we work on Glushkov graphs representing regular expressions E in the schema: this representation gives straightforwardly the right places in E that may be changed while preserving validity.*

1. Introduction

More and more applications use XML to store and exchange their data, which are often stored in XML databases. In this context, XML documents are valid, *i.e.*, they respect a schema. The schema serves two purposes: (i) it defines an interface for programs and users to query the data, and (ii) it determines how the database management system physically stores the data on the disk. In an XML environment it is quite natural to have the ability to respond to changes in the real world by allowing the schema to evolve, especially within the Web framework.

The schema evolution problem deals with the update of a schema when it no longer meets the needs of the user. The goal of schema evolution research is to allow schemas to change while maintaining access to the existing data. Indeed, the evolution can be *conservative* or *non-conservative*. In the first case, all documents that were valid w.r.t. the old schema are valid w.r.t. the new one. In the second case, documents valid for the original schema are no more guaranteed to meet the structural constraints described by the evolved schema.

The non-conservative schema evolution may be problematic since it is necessary to validate all documents against the new schema and, if they are not valid, they should be adapted to it. The document adaption process can provoke data loss since it may be necessary to delete tags (and their information) from it. Moreover, when documents to be revalidated are stored in different sites, not only their transfer cost should be considered (in addition to the whole revalidation cost), but also problems due to access control should be faced.

To our knowledge, most of work in schema evolution deals with a non-conservative approach. Indeed, the focus has been first to define schema update primitives: in [Su et al. 2001], a complete and sound set of primitives is proposed, which can make previously valid documents invalid. In this case, the authors propose changes to be performed on documents in order to make them valid. The same approach is followed in [Al-Jadir and El-Moukaddem 2003]. For example, their *change the parent relationship* primitive may change an element occurrence from n times to exactly once. In this case, all documents having this element repeated must be changed in order to have it only once. In [Prashant and Kumar 2006], the authors try to solve the problem of revalidation by building an XSLT script to force the document to be valid with relation to the new schema (as an extension of the approach proposed in [Su et al. 2001]). In [Guerrini et al. 2005, Mesiti et al. 2006], a set of schema update primitives is also proposed and the impact of schema updates over documents is analysed. In the same way as in [Raghavachari and Shmueli 2004], the authors take advantages of similarities and differences between the old schema and the new one to avoid validating portions of documents. The basic idea is, considering the updates made to the schema, to identify the parts of the new schema that would require that documents must be revalidated. Only the document portions corresponding to these schema parts are then revalidated (and are changed, if necessary).

From above, we can conclude that, although the conservative evolution of schema has been identified as a desirable feature for XML databases [Roddick et al. 2000, Costello and Schneider 2000], there is not a significant amount of research work in this area. The approach proposed in [Bouchou et al. 2004] is conservative, however the new schema is inferred from an invalid document, that is, the authors do not consider schema update primitives.

In this paper, we consider a framework for schema evolution such as the one proposed in [Guerrini et al. 2005, Mesiti et al. 2006], and we present a way of assisting user to specify schema updates with no impact on validity. To do so, we propose intuitive way to enter updates, together with a method to determine how to implement them as conservative extensions of the original schema (*i.e.* keeping existing documents valid without making any change on them).

In this way, we try to respond to the increasing demand for tools specially designed for administrators not belonging to the computer science community, but capable to make decisions on the evolution of an application [Roddick et al. 2000]. This kind of user needs a system that assures a consistent evolution of the schema in an incremental way.

To illustrate this need, let us suppose that a librarian is responsible for feeding an XML database with information about laboratories and their publications. Suppose also that the current XML schema accepts only journal articles, while the laboratories want conference articles to be also stored as publications in the database. Thus, the librarian receiving this demand has to change the schema accordingly. In this situation, the librarian is not a computer science expert, thus he/she should have a tool to assist him/her to evolve the schema.

We consider an XML schema as a set of rules. Each rule uses a regular expression to define the allowed sub-elements of an element. The modifications on an XML schema consist in changes on the regular expressions of the schema. Indeed, our algorithm is based on the computation of a new regular expression to extend a given regular language in a conservative way. Thus, our problem can be formulated in terms of regular expression evolution:

Given a regular expression E, suppose that an update must be performed over E. A new regular expression E' is built from E such that $L(E) \subseteq L(E')$.

The following example gives an overall idea of the method.

Example 1.1 Consider a schema $\mathcal{S}$ that constraints XML documents storing researchers and their publications. Suppose that one of the constraints in $\mathcal{S}$ is: publications are grouped by journal articles organized by subject and year of publication. That is, the content model of element $Publication$ is modeled by the regular expression E[1]=$Subject\ (Year\ Journal^{+})^{*}$. Let the following extract of an XML document d valid with respect to S:

<**Publication**>
  <**Subject**> Automata </**Subject**>
  <**Year**> 1965 </**Year**>
  <**Journal**> Theorical Computer Science </**Journal**>
  <**Journal**> International Journal of Computer Science </**Journal**>
  ⋮
</**Publication**>

Supposing that a user wants to update S by inserting a new element $Conference$ into the content model of $Publication$ (*i.e.*, $Subject\ (Year\ Journal^{+})^{*}$), the resulting regular expression E' may be $Subject\ (Year\ Journal^{+}\ \texttt{Conference}^{+})^{*}$, and, in this case, d is no more valid with respect to the new schema $\mathcal{S}'$, since the new element was inserted as a mandatory element, in sequence with $Journal^{+}$. This example shows that the user must know the syntax of regular expressions to be able to update the schema otherwise he/she may invalidate all the database.
In this situation, if the user could insert the new element by just *saying* that the element must follow element $Journal$, that it must be at the same level as element $Journal$, and *that he/she wants the update to preserve the validity*, then our system would build, for example, the following regular expression $E' = Subject\ (Year\ (Journal|\texttt{Conference})^{+})^{*}$. In this case d still is valid with respect to both $\mathcal{S}'$ and $\mathcal{S}$. $\square$

The Example 1.1 shows how difficult can be the schema evolution: the user should be aware that elements order must be respected and that the new element should be inserted as non-mandatory, otherwise the documents will become invalid with respect to the new schema. Indeed, expressions like $Subject\ (Year\ (Journal|Conference)^{+})^{*}$ are not trivial to build from $Subject\ (Year\ Journal^{+})^{*}$. More generally, the task of finding the places where the new symbol may be added is not trivial.

Our idea is to allow the user to update the content model of an element without worrying about details from the regular expression that describes this content model. Notice that this can be achieved only for a subset of update primitives. Indeed, if the update is the removal (or the replacement) of a mandatory sub-element, it is unavoidable to delete (or replace) this sub-element in all documents where it appears, otherwise the documents will become invalid. Nevertheless, as shown in [Guerrini et al. 2005], some update primitives are known to have no impact on validity under certain conditions. These primitives are the following:

[1]Most of XML schemas are modeled by regular expressions [L. and Chu 2000].

- Sub-element insertion in a content model.
- Cardinality extension of a sub-element in a content model: for instance making a sub-element optional, or allowing repetition of an existing sub-element.
- Substructure (regular expression) insertion in a content model.
- Element creation: a new rule is added to schema to describe a new element (probably inserted as a sub-element in another rule).

In this paper, we propose a set of primitives among those which are able to have no impact on validity. In addition, we aim to hide from the user the conditions for keeping the consistency of the XML database. For example, the user may insert a sub-element e in a content model c by just $saying$ that e must be inserted into c, with several options such as the context, the type (sequence or choice), etc. The resulting content model c' will always have e as an optional sub-element. To summarize, we consider a framework in which the user can specify one update in an intuitive way, and ask that this update must preserve the validity of existing documents, and we propose a method to compute the accurate update.

In what follows we first present the theoretical context of our proposition (section 2), then we describe our method (section 3), *i.e.* the update primitives and their implementation as a conservative schema evolution.

2. Background

We first present the schema model used in this paper, then we describe the notions used in our schema evolution assistance method.

We remind that XML documents are seen as unranked labeled trees (*i.e.*, trees whose nodes have a finite but arbitrary number of children) and, consequently, XML schemas are modeled as regular tree grammars. More precisely, we define the schema model (based on [Papakonstantinou and Vianu 2000]) as follows.

Definition 2.1 A schema $\mathcal{S}$ over an alphabet Σ for XML documents consists of a root type in Σ and a mapping associating to each $a \in \Sigma$ a language over Σ. The language associated to a is described by a regular expression E_a. We call *content model* of a the regular expression E_a associated to a. □

Example 2.1 Suppose a schema that models the document presented in Example 1.1. The schema $\mathcal{S}$ is, therefore, as follows:

$\Sigma = \{University, Lab, Name, Members, Publications, Position, Subject, Year, Journal, Title, text\}$

$root : University$

$University : Lab^*$ (1)

$Lab : Name\ Members^+\ Publication^*$ (2)

$Publication : Subject\ (Year\ Journal^+)^*$ (3)

$Members : Position\ Name$[2] (4)

Notice that, for example, the rule (3) represents the content model of $Publication$ which is modeled by the regular expression $E_{Pub} = Subject(Year\ Journal^+)^*$ and the intended semantics of E_{Pub} is that the production of a given author is stated by the area (or subject) of his/her research, followed by a list of journal papers, presented by year. □

[2]The other rules have the form $e :\ text$ or $e :\ \varepsilon$, where $text$ represents a text value and ε represents the empty word.

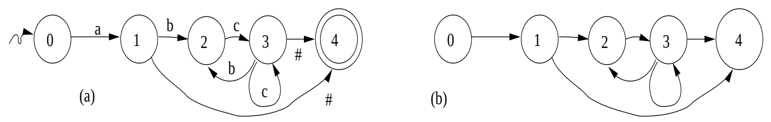


Figure 1. (a) Glushkov Automaton for $a(b\ c^+)^*\#$. (b) Its Glushkov graph

For the sake of simplicity, we adopt here the simplest schema model, corresponding to a DTD. But our method can be used for every schema language based on regular expressions (for a taxonomy of such languages see [Murata et al. 2005]). Notice also that we do not consider XML attributes since they are not constrained by regular expressions: possibilities of conservative change concerning attribute definition are limited since one can only add non compulsory attributes.

Now we introduce a method to transform a graph built from a finite state automaton (FSA) into its corresponding regular expression. This method, which is the basis of our approach to update conservatively XML schemas, is outlined as follows: starting from regular expressions defining the content models of elements, we build Glushkov automata based on the algorithm proposed in [Bruggeman-Klein 1993], which is known to be efficient. These automata are seen as directed graphs and are stored with the corresponding content models. A schema update is then performed over the graph that corresponds to the content model to be updated. After that, the changed graph is reduced to a regular expression following the method proposed in [Caron and Ziadi 2000] (which is, again, very efficient).

Our motivation to use the reduction process is that it allows to identify the starred sub-expressions of a regular expression E and, thus, it is quite simple to introduce updates in right places in order to be sure to extend the language in a conservative way. In the following we detail the reduction method and we define the notions of *orbits* and *contexts* that are need to identify the starred sub-expressions.

A Glushkov automaton is built by subscripting each symbol in the regular expression with its position. In this way, the automaton is homogeneous[3].

Example 2.2 Given the regular expression[4] $E = (a(b\ c^+)^*\#$, the subscripted regular expression is $\overline{E} = a_1(b_2\ c_3{}^+)^*\#_4$. The Glushkov automaton $M = (\Sigma, Q, \Delta, q_0, F)$, built from E, is such that: the alphabet is $\Sigma = \{a, b, c, \#\}$, the set of states is $Q = \{0, 1, 2, 3, 4\}$, the initial state is $q_0 = 0$, the set of final states is $F = \{4\}$ and the transition relation Δ is defined by the edges of the graph in Figure 1(a). □

A *Glushkov graph* is the directed graph $\mathcal{G} = (X, U)$ obtained from a Glushkov automaton such that each node in X corresponds to a state and each edge in U to a transition. Since Glushkov automata are homogeneous, the edges of a $\mathcal{G}$ are not labeled as shown in Figure 1(b). A graph has a *root* node r (resp. an *antiroot* s) if there exists a path from r to any node in the graph (resp. from any node in the graph to s). A graph is a *hammock* if it has both a root (r) and an antiroot (s), with $r \neq s$. Thanks to the end mark (#), the Glushkov graphs used in this work are *hammocks*.

Now we consider the graph properties taken from [Caron and Ziadi 2000], that

[3]A FSA is said to be *homogeneous* [Caron and Ziadi 2000] if one always enters a given state by the same symbol.

[4]We add an end mark ($\#$) to the regular expression E and to every string belonging to $L(E)$.

will be used later on in this work. A set $\mathcal{O} \subseteq X$ is said to be an *orbit* if for all x and x' in $\mathcal{O}$ there exists a non trivial path from x to x'. An orbit is *maximal* if for each node x of $\mathcal{O}$ and for each node x' out of $\mathcal{O}$, there does not exist a path from x to x' and a path from x' to x. Notice that an orbit is maximal if it is not contained in any other orbit. Let $\mathcal{O}$ be an orbit, we define: $In(\mathcal{O})$=$\{x \in \mathcal{O} \mid \exists x' \in (X \setminus \mathcal{O}), (x', x) \in U\}$ as the *input* of $\mathcal{O}$ and $Out(\mathcal{O})$=$\{x \in \mathcal{O} \mid \exists x' \in (X \setminus \mathcal{O}), (x, x') \in U\}$ as the *output* of $\mathcal{O}$. An orbit $\mathcal{O}$ is *stable* if $\forall x \in Out(\mathcal{O})$ and $\forall y \in In(\mathcal{O})$, the edge (x, y) exists. An orbit $\mathcal{O}$ is *transverse* if $\forall x, y \in Out(\mathcal{O})$, $\forall z \in (X \setminus \mathcal{O}), (x, z) \in U \Rightarrow (y, z) \in U$, and if $\forall x, y \in In(\mathcal{O}), \forall z \in (X \setminus \mathcal{O}), (z, x) \in U \Rightarrow (z, y) \in U$. An orbit $\mathcal{O}$ is *strongly stable* (resp. *strongly transverse*) if it is stable (resp. transverse) and if after deleting the edges in $Out(\mathcal{O}) \times In(\mathcal{O})$, every suborbit is strongly stable (resp. strongly transverse).

Example 2.3 The graph (a hammock) of Figure 1(b) has 2 orbits. The orbit $\mathcal{O}_1 = \{2, 3\}$, with $In(\mathcal{O}_1) = \{2\}$ and $Out(\mathcal{O}_1) = \{3\}$, is maximal. Orbit $\mathcal{O}_2 = \{3\}$ is not maximal. Orbit $\mathcal{O}_2$ is stable since all the edges in $Out(\mathcal{O}_2) \times In(\mathcal{O}_2)$ are in $\mathcal{O}_2$. It is transverse since the edge $(3, 4)$ exists in the graph. In fact, $\mathcal{O}_1$, $\mathcal{O}_2$ are strongly stable and strongly transverse. $\square$

Given a graph $\mathcal{G}$ in which all orbits are strongly stable, we build a *graph without orbits* $\mathcal{G}_{wo}$ by recursively deleting, for each maximal orbit $\mathcal{O}$, all edges (x, y) such that $x \in Out(\mathcal{O})$ and $y \in In(\mathcal{O})$. The process ends when there are no more orbits. Notice that $\mathcal{G}_{wo}$ is defined in a unique way [Caron and Ziadi 2000]. During the construction of $\mathcal{G}_{wo}$ a structure that stores the computed maximal orbits of $\mathcal{G}$ is also built. This structure is called *hierarchy of orbits*. The hierarchy of orbits $\mathcal{H}$ is hierarchically organized according to the set-inclusion relation.

Given $\mathcal{G}_{wo}$, it is said to be *reducible* ([Caron and Ziadi 2000]) if it is possible to reduce it to one state by successive applications of any of the three rules $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{R}_3$ explained below (illustrated by Figure 2). Let x be a node in $\mathcal{G}_{wo} = (X, U)$. We note $Q^-(x) = \{y \in X \mid (y, x) \in U\}$ the set of immediate predecessors of x and $Q^+(x) = \{y \in X \mid (x, y) \in U\}$ the set of immediate successors of x. The reduction rules are defined as follows (we denote $r(x)$ the regular expression associated to node x, and e the resulting regular expression in each case):

Rule $\mathbf{R}_1$: If two nodes x and y are such that $Q^-(y) = \{x\}$ and $Q^+(x) = \{y\}$, then replace node x by node xy and delete node y.

Rule $\mathbf{R}_2$: If two nodes x and y are such that $Q^-(x) = Q^-(y)$ and $Q^+(x) = Q^+(y)$, then replace node x by node $x|y$ and delete node y.

Rule $\mathbf{R}_3$: If a node x is such that $y \in Q^-(x) \Rightarrow Q^+(x) \subset Q^+(y)$, *i.e.*, each predecessor of node x is also a predecessor of any successor of node x, then delete the edges going from $Q^-(x)$ to $Q^+(x)$. In this case we build a regular expression in the following way: if $r(x)$ is of the form E (resp. E^+) then e will be $E?$ (resp. E^*).

The reduction process starts at the lowest level of the hierarchy of orbits and works bottom-up, from the smallest orbits to the maximal ones (set inclusion). Indeed, during the construction of $\mathcal{G}_{wo}$, the orbits are hierarchically ordered, according to the set-inclusion relation. The information concerning the orbits of the original graph is used to add the transitive closure operator ("$^+$") to the regular expression being constructed. Thus, during the reduction process, when a single node representing a whole orbit is obtained, its

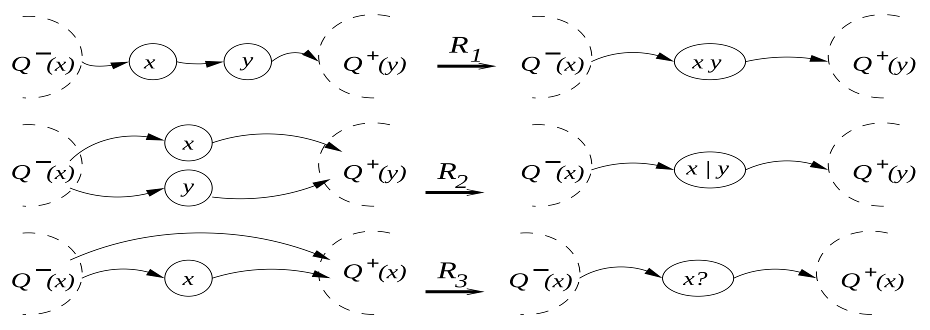


Figure 2. Rules R$_1$, R$_2$ and R$_3$

content is decorated with a “$^+$”.

Example 2.4 Figure 3 shows some steps of reduction of the Glushkov graph $\mathcal{G}$ that represents the regular expression $E = (a(b|c)^*)^*\#$ ($\overline{E} = (a_1(b_2|c_3)^*)^*\#_4$): ($i$) starting from $\mathcal{G}$ without orbits (Figure 3(a)), (ii) Rule **R**$_2$ is applied over nodes 2 and 3 (Figure 3(b)), (iii) node $2|3$ is decorated with “+” since it represents a whole orbit (Figure 3(c)), (iv) Rule **R**$_3$ is applied over node $(2|3)^+$ resulting in $(2|3)^*$ (Figure 3(d)), (v) Rule **R**$_1$ is applied over nodes 1 and $(2|3)^*$ and the resulting node is decorated with “+” since it represents a whole orbit (Figure 3(e)), and (vi) Rule **R**$_3$ is applied over node $(1(2|3)^*)^+$ resulting in $(1(2|3)^*)^*$ (Figure 3(f)). This process continues up to the original graph is reduced to just one node containing the positional regular expression, *i.e.*, $0(1(2|3)^*)^*4$. Discarding the position 0, it corresponds to the original regular expression $(a(b|c)^*)^*\#$. □

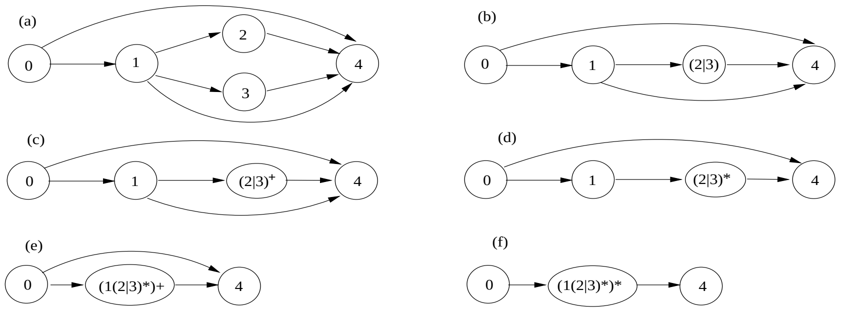


Figure 3. Reduction of a Glushkov graph without orbits built from $(a(b|c)^*)^*\#$

A characterization of Glushkov automata is given by the following theorem.

Theorem 2.1 *[Caron and Ziadi 2000] A finite state automaton M is a Glushkov automaton iff its graph $\mathcal{G} = (X, U)$ satisfies the following conditions:* (1) *$\mathcal{G}$ is a hammock,* (2) *each maximal orbit in $\mathcal{G}$ is strongly stable and strongly transverse and* (3) *the graph without orbit of $\mathcal{G}$ is reducible. In this case, $\mathcal{G}$ is called a Glushkov graph.*

It has been shown in [Bruggeman-Klein and Wood 1992] that every maximal orbits of a Glushkov graph $\mathcal{G}$ built from a regular expression E represent starred subexpressions of E.

Example 2.5 Giving the regular expression $E = a(b\ c^+)^*\#$ (and $\overline{E} = a_1(b_2\ c_3^+)^*\#_4$). From the graph $\mathcal{G}$ of Figure 1(b), a graph without orbits $\mathcal{G}_{wo}$ is built and we have $\mathcal{H} = \{\{2,3\},\{3\}\}$. The orbit $\{3\}$ represents the starred subexpression c^+ and the orbit $\{2,3\}$ represents the starred subexpression $(b\ c^+)^*$. □

We use the hierarchy of orbits $\mathcal{H}$ of a Glushkov graph to define the contexts in a regular expression. For each orbit in $\mathcal{H}$, a context is defined. Moreover, we define also a context called *general* having all symbols that do not participate in any orbit of $\mathcal{H}$.

Definition 2.2 Let E be a regular expression and $\overline{E}$ be the subscripted regular expression of E. Let M_E be a Glushkov automaton built from $\overline{E}$ and $\mathcal{G} = (X, U)$ be the graph built from M_E. Let $\mathcal{H}$ be the hierarchy of orbits of $\mathcal{G}$. Let $\chi(p)$ denote the symbol in E that corresponds to each position p in $\overline{E}$ (remind that each position p in $\overline{E}$ is also a node n in X). The contexts $\mathcal{C}$ of E are defined as follows:
(i) for each orbit $\mathcal{O} \in \mathcal{H}$, $\mathcal{C}_{\mathcal{O}} = \{\chi(p) \mid \ p \in \mathcal{O} \ \wedge \forall O_1 \subset \mathcal{O} \ p \notin \mathcal{O}_1\}$
(ii) the general context is: $\mathcal{C}_{general} = \{\chi(p) \mid \ \forall \mathcal{O} \in \mathcal{H} \ \Rightarrow p \notin \mathcal{O}\}$. □

We notice that if a regular expression E has no starred sub-expression, the only context of E is the general one. On the other hand, if all symbols of E are in starred sub-expressions, the general context is empty.

Example 2.6 Giving the regular expression $E = a(b\ c^+)^*\#$ and the hierarchy of orbits $\mathcal{H} = \{\{2,3\},\{3\}\}$ from Example 2.5. The built contexts are $\mathcal{C}_{\{3\}} = \{c\}$, $\mathcal{C}_{\{2,3\}} = \{b\}$ and $\mathcal{C}_{general} = \{a\}$. □

3. Schema Evolution Framework

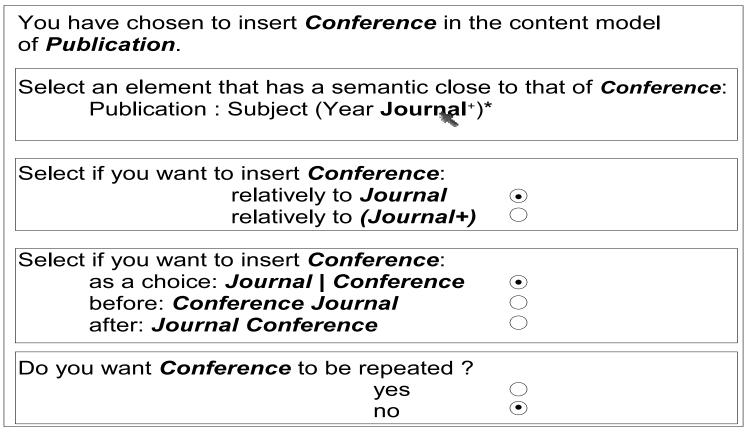
3.1. Update Primitives

In our approach the update primitives specified by the user are executed on Glushkov graphs that represent regular expressions. The use of Glushkov graphs to implement these primitives allows our method to utilize the orbits (*i.e.*, the starred sub-expression in E) to guide the way that E is updated.

We propose four atomic primitives for the creation of a new element, the insertion of a sub-element in a content model, the extension of the cardinality of a sub-element (from *at most once* to *several*) and the possibility to make a mandatory sub-element optional. These atomic primitives are based on the work presented in [Guerrini et al. 2005].

These primitives may be directly written as such, but, since our aim is to assist the user in updating the schema, we propose an interactive tool that allows the user to specify each feature of the update query in an intuitive way and, then, to built the correspondingly primitive. Among these features (*e.g.*, an insertion), we must have the *context* in which the update is expected to appear. We propose to denote it by labels appearing in starred sub-expressions (see Example 2.6). More precisely, the user gives one element name to identify the target starred sub-expression s, then he/she chooses whether the inserted sub-element has to appear before or after s. Notice that the user may choose a label that appears more than once in the regular expression: in this case, the system should obtain the accurate occurrence, *i.e.*, the right subscript in the subscripted expression.

We consider the Glushkov graph $\mathcal{G} = (X, U)$ built from a regular expression E that describes the content model of the element to be updated. We remind that in $\mathcal{G}$ each node (but the node 0) corresponds to a position in the subscripted regular expression $\overline{E}$. The only node that does not have out edges is subscripted with the position of the end mark (#). By abuse of notation, we use both *context* and *orbits* interchangeably. Thus, when referring to the inputs of a context $\mathcal{C}$ we mean the symbols that correspond to the positions belonging to the inputs of the orbit $\mathcal{O}$ from which $\mathcal{C}$ was built (see Definition 2.2).



You have chosen to insert ***Conference*** in the content model of ***Publication***.

Select an element that has a semantic close to that of ***Conference***:
Publication : Subject (Year **Journal**$^{+}$)*

Select if you want to insert ***Conference***:
[✓ relatively to ***Journal***]
relatively to ***(Journal+)***

Select if you want to insert ***Conference***:
[✓ as a choice: ***Journal | Conference***]
before: ***Conference Journal***
after: ***Journal Conference***

Do you want ***Conference*** to be repeated ?
yes
[✓ no]

Figure 4. Steps of the interaction for an insertion

Our schema update primitives work on the Glushkov graph, by inserting edges (also a node for the insertion primitive). Indeed, as the schema update must keep the validity of documents, no deletion is performed on the graph (neither node deletion, nor edge deletion). Once modified, the Glushkov graph is reduced to its corresponding regular expression E' and E' replaces E in the original content model. Graph modifications are performed as follows:

- Insertion: the node n (that represents the element e to be inserted into E) is inserted into the corresponding Glushkov graph $\mathcal{G}$ (in a given position or context), and new edges are created from/to n.
- Making an element optional: given a node $n \in X$ that represents a mandatory element e in E, $\mathcal{G}$ is updated as follows: edges are added from predecessors of n to successors of n.
- Extending the cardinality of an element: given a node $n \in X$ that represents an element e in E, $\mathcal{G}$ is updated as follows: a new singleton orbit $\mathcal{O}$ containing only the position of e is inserted in the hierarchy $\mathcal{H}$ of $\mathcal{G}$.
- Creation: given a new element e and its content model E_e, a new rule $e : E_e$ is inserted into the schema $\mathcal{S}$.

The atomic primitives can be composed to form high level primitives in order to express more complex updates in a more compact way. For example, it is possible to insert an expression into a content model or to make optional a whole sub-expression.

3.2. Insertion Primitive

The advantage of our proposition is to allow a user to express his/her need in an intuitive way, while guaranteeing that the schema update will keep the existing document validity. In order to illustrate how the user can specify an insertion, we show in Figure 4 main steps of the interaction. First, the user has chosen the insertion operation, the content model to be updated (element $Publication$), and he/she has given the name of the element to be inserted ($Conference$). After that, the user can choose one element in the regular expression by clicking on it: in Figure 4, he/she has chosen $Journal$. Then it is possible to choose either the element or its context as the reference of the insertion: by default the element is chosen (*i.e.*, $Journal$ is chosen, and not $Journal^{+}$). It is also necessary to precise the relationship between the new element and the chosen reference: in Figure 4, *choice* has been chosen. Last, the user is asked whether the new element may be repeated or not: by default it is not repeated.

The update operation corresponding to the choices in Figure 4 is the following: $ins(Publication, Conference, 3, false, choice, false)$. Notice that, since the user selects one element in the expression E, and only one, it is possible to get the corresponding position in $\overline{E}$ without ambiguity (in the previous example, the user chose $Journal$, so the generated update operation has the position 3 as third parameter). We define now the insertion atomic primitive.

Definition 3.1 Let e be an element of a schema $\mathcal{S}$ (*i.e.*, $e \in \Sigma$) and E the content model of e. Let e' the new element e' to be inserted. Let τ be a position in $\overline{E}$, associated to the element in E, which is the reference for the insertion. Let $\mathcal{G}$ be a Glushkov graph built from E, and let $\mathcal{C}$ be the context built from an orbit $\mathcal{O}$ (of $\mathcal{G}$) such that $\tau \in \mathcal{O}$. We denote by $ins(e,\ e',\ \tau,\ context,\ mode,\ times)$ the atomic primitive for inserting an element e' into the content model of e, where:

- $context$ is a boolean that defines the reference of insertion, *i.e.*, whether e' is inserted relatively to $\mathcal{C}$ ($context$=$true$) or τ ($context$=$false$),
- $mode$ defines whether e' is inserted as a *choice*, a $sequence\text{-}before$, or a $sequence\text{-}after$, in relation to the chosen reference, and
- $times$ is a boolean: if $times$=$true$ then e' will be decorated with $^+$. □

Notice that the insertion of e' in E is guided by the most internal orbit where τ appears.

Example 3.1 The example of interaction in Figure 4 leads to the update $ins(Publication, Conference, 3, false, choice, false)$, which will result in the regular expression E': $Subject\ (Year\ (Journal \mid Conference)^+)^*\#$. Remark that the contexts are $\mathcal{C}_{\{3\}}$=$\{Journal\}$, $\mathcal{C}_{\{2,3\}}$=$\{Year\}$ and $\mathcal{C}_{general}$=$\{Subject\}$. □

Figure 5 presents the algorithm ins, which implements the insertion primitive. First, a new node $n_{e'}$ representing the element to be inserted is added to the graph (line 08). Next, new edges are inserted in $\mathcal{G}$ depending on the input parameters (lines 09 - 23): the first test is whether the new element will be inserted as a $choice$, a $sequence\text{-}before$ or a $sequence\text{-}after$, and next whether the reference is the position τ or its orbit. Before building the new regular expression (line 27), the orbit $\mathcal{O}'$ and all orbits in which $\mathcal{O}'$ is included are updated: $n_{e'}$ is inserted and the inputs and outputs are also updated (line 24). It is also tested whether the input parameter $times$ is true: if it is the case, a singleton orbit is inserted into $\mathcal{H}$ (line 25). This new orbit will provoke the decoration of node $n_{e'}$ with "+", which is transformed into "*" by the reduction rules (line 26) since $n_{e'}$ is inserted as an optional node.

Figure 6 shows the graph built by Algorithm ins (Figure 5) from the Example 3.1:

• As the $context$ parameter is $false$, the new node 5 is inserted into the orbit where c appears, *i.e.*, $\mathcal{O} = \{3\}$, in relation with c (*i.e.*, $\tau = 3$).

• As the $mode$ parameter is $choice$, all successors of node 3 have edges from the new node 5 (line 12 of Figure 5): $(5, 4)$. Also, all predecessors of 3 have edges to the new node (line 13 of Figure 5): $(2, 5)$.

• $\mathcal{O}$ and the orbits in which $\mathcal{O}$ is included are updated with the new node (line 27 of Figure 5). Thus, the new hierarchy of orbits is $\{\{3, 5\}, \{2, 3, 5\}, \{0, 1, 2, 3, 4, 5\}\}$.

• Finally, a new regular expression is built from $\mathcal{G}_{wo}$ (line 29 of Figure 5).

```
Algorithm $ins(e, e', \tau, context, mode, times)$
Output:  The new content model $E'$ of $e$.
01.  Let $\mathcal{S}$ be the schema.
02.  Let $e : E_e$ in $\mathcal{S}$ be the representation of the content model of $e$.
03.  Let $\mathcal{G} = (X,U)$ be a Glushkov graph built from $E_e$.
04.  Let $\mathcal{H}$ be the hierarchy of orbits of $\mathcal{G}$.
05.  Let $\mathcal{O}$ be the orbit in $\mathcal{H}$ such that $\tau \in \mathcal{O}$.
06.  Let $n_{e'}$ be the new node, representing $e'$ ($n_{e'} \notin X$)
07.  $\mathcal{G}' \leftarrow \mathcal{G}$; $\mathcal{H}' \leftarrow \mathcal{H}$
08.  Insert $n_{e'}$ into $X'$
09.  If $mode = choice$
10.      If $context$ Insert $(n_{e'}, n_o)$ into $U'$ for each $n_o \in Q^+(Out(\mathcal{O}'))$
11.                   Insert $(n_i, n_{e'})$ into $U'$ for each $n_i \in Q^-(In(\mathcal{O}'))$
12.      Else Insert $(n_{e'}, n_o)$ into $U'$ for each $n_o \in Q^+(\tau)$
13.             Insert $(n_i, n_{e'})$ into $U'$ for each $n_i \in Q^-(\tau)$
14.  Else If $mode = sequence\text{-}before$
15.      If $context$ Insert $(n_{e'}, n)$ into $U'$ for each $n \in Q^-(In(\mathcal{O}'))$
16.                   Insert $(n, n_{e'})$ into $U'$ for each $n \in (In(\mathcal{O}')$
17.      Else Insert $(n_{e'}, \tau)$ into $U'$
18.             Insert $(n, n_{e'})$ into $U'$ for each $n \in Q^-(\tau)$
19.  Else If $mode = sequence\text{-}after$
20.      If $context$ Insert $(n_{e'}, n)$ into $U'$ for each $n \in (Out(\mathcal{O}')$
21.                   Insert $(n, n_{e'})$ into $U'$ for each $n \in Q^+(Out(\mathcal{O}'))$
22.      Else Insert $(n_{e'}, n)$ into $U'$ for each $n \in Q^-(\tau)$
23.             Insert $(\tau, n_{e'})$ into $U'$
24.  Update $\mathcal{O}'$ and $\mathcal{O}'_i$ with node $n_{e'}$, for each $\mathcal{O}'_i \supset \mathcal{O}'$
25.  If $times$ Insert $\{n_{e'}\}$ into $\mathcal{H}'$
26.  $\overline{E'} \leftarrow$reduce($\mathcal{G}'$, $\mathcal{H}'$)
27.  Translate $\overline{E'}$ into $E'$ using $\chi$, with $\chi(n_{e'}) = e'$
```

Figure 5. Algorithm that implements the insertion operation.

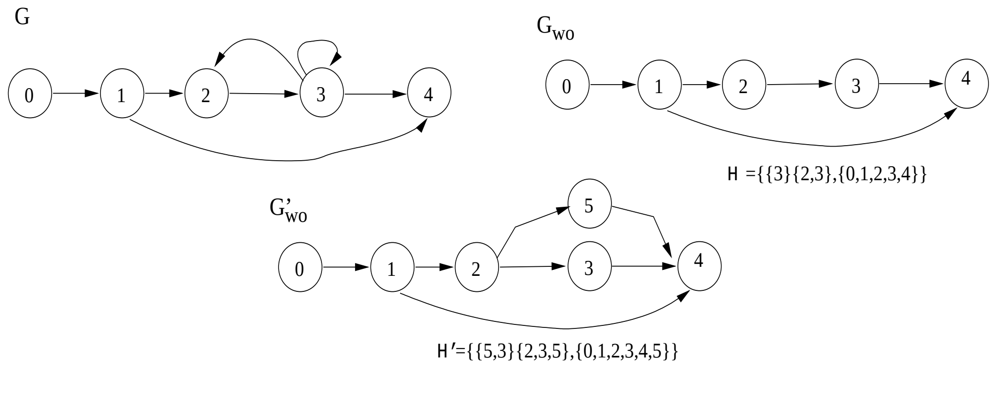


Figure 6. Operation $ins(Publication,\ Conference,\ 3,\ false,\ choice,\ false)$ **applied over** $\mathcal{G}_{wo}$ **given** $\mathcal{G}'_{wo}$

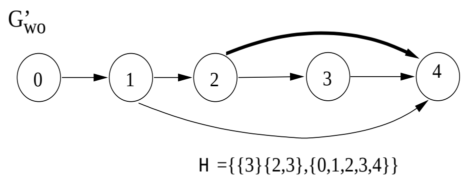


Figure 7. Operation $makeOpt(y, c)$ applied over $\mathcal{G}_{wo}$ (Figure 6).

3.3. Other Primitives

We define the atomic primitive to make optional an element (thus, permitting to delete occurrences of this element in documents while keeping them valid).

Definition 3.2 Let e be an element of a schema $\mathcal{S}$ (*i.e.*, $e \in \Sigma$) and E be the content model of e. Let e' be a mandatory sub-element in E, such that τ is its position. Let $\mathcal{G}$ be a Glushkov graph built from E. We denote by $makeOpt(e, \tau)$ the atomic primitive, which makes e' optional. This operation is performed in $\mathcal{G}$ as follows: all predecessors of τ will have edges to all successors of e'. The orbit where e' appears is updated if necessary. $\square$

Example 3.2 Given the regular expression $E_y = a(b\ c^+)^*\#$. For the operation $makeOpt(y, 3)$ the resulting regular expression E' is $a(b\ c^*)^*\#$.
In order to compute E', we modify the Glushkov graph associated to E. Figure 7 presents the resulting graph $\mathcal{G}'_{wo}$ after executing $makeOpt(y, 3)$ over the graph $\mathcal{G}_{wo}$ from Figure 6: new edges are inserted into $\mathcal{G}_{wo}$, from the predecessors of node 3 (node that represents element c) to the successors of 3. Thus, the edge $(2, 4)$ is inserted, rending optional the node 3. Remark again that the regular language described by E' includes the regular language described by E, *i.e.*, $L(E) \subseteq L(E')$. $\square$

Another primitive operation is to extend the cardinality of an element, from *only once* to *several times*.

Definition 3.3 Let e be an element of a schema $\mathcal{S}$ (*i.e.*, $e \in \Sigma$) and E be its content model. Let e' be a sub-element in E at position τ. Let $\mathcal{G}$ be a Glushkov graph built from E. Let $\mathcal{H}$ be the hierarchy of orbits of $\mathcal{G}$.
We denote by $ExtendCard(e, \tau)$ the atomic primitive, which extends the cardinality of e'. The operation is performed as follows: a new singleton orbit $\{\tau\}$ is inserted into $\mathcal{H}$, respecting the set-inclusion relation. $\square$

Notice that if τ is optional, the reduction process decorates τ with a "*", otherwise with a "+", in $\overline{E'}$. Then $\overline{E'}$ is translated into E' using χ, with $\chi(\tau) = e'$.

The last primitive operation allows to build a content model for a new element in the schema. The following definition formalizes it.

Definition 3.4 Let e be an element name and E be the content model of e. We denote by $createCM(e, E)$ the atomic primitive to add the rule $e : E$ in $\mathcal{S}$ (and, if necessary, $e \in \Sigma$). If $e \in \Sigma$ and has already a content model E_e in $\mathcal{S}$, then the new rule in $\mathcal{S}$ will be: $e : E_e|E$. $\square$

The execution of all the atomic primitives first verifies some simple preconditions: (i) to make an element e' optional or to extend its cardinality in a given content model E, e' must belong to E, (ii) to create a new rule $e : E$, the rule $e : E$ should not exist already in $\mathcal{S}$, and (iii) to extend the cardinality of an element e', e' must not be already decorated with "*" or "$^+$". For each atomic primitive, the time complexity is constant, as the Glushkov graph representations are built once (and stored).

We can notice that each of our atomic primitives applied on a schema S that conforms to Definition 2.1 outputs a schema S' that still conforms to Definition 2.1. Moreover, the following theorem states that the graph $\mathcal{G}'_{wo}$ built by applying Definitions 3.1, 3.2 and 3.3 can be reduced to a regular expression (the new content model).

Theorem 3.1 *Let $\mathcal{G}$ be a Glushkov graph, $\mathcal{G}_{wo}$ its graph without orbits and $\mathcal{H}$ the hierarchy of orbits of $\mathcal{G}_{wo}$. The graph $\mathcal{G}'$ (i.e., $\mathcal{G}'_{wo}$ and $\mathcal{H}'$) built from $\mathcal{G}_{wo}$ and $\mathcal{H}$ by applying Definitions 3.1, 3.2 or 3.3 still be a Glushkov graph, in particular $\mathcal{G}'_{wo}$ is reducible by successive applications of rules $\mathbf{R}_i$.* □

Proof (Sketch): Our assumption is that $\mathcal{G}$ is a Glushkov graph (cf. Theorem 2.1: *(1)* $\mathcal{G}$ is a hammock, *(2)* each maximal orbit in $\mathcal{G}$ is strongly stable and strongly transverse and *(3)* the graph without orbit of $\mathcal{G}$ is reducible). For each atomic primitive, both its definition (3.1 to 3.3) and its implementation are designed so that a node and/or edges are inserted, in such a way that the reduction conditions are respected, thus, $\mathcal{G}'_{wo}$ is reducible (*i.e.*, Rules $\mathbf{R}_1$, $\mathbf{R}_2$ or $\mathbf{R}_3$ can be applied over $\mathcal{G}'_{wo}$). Moreover, the hierarchy of orbits $\mathcal{H}$ is also updated respecting the set-inclusion relation, so $\mathcal{H}'$ still is strongly stable and strongly transverse, and $\mathcal{G}'$ still is a hammock as neither initial state nor final state is added. □

Finally, the following theorem states that the content models built by our atomic primitives are consistency-preserving, that is, the XML documents valid with respect to the original schema are still valid with respect to the new schema.

Theorem 3.2 *Let $\mathcal{S}$ be an XML schema. Let $e : E_e$ be a rule in $\mathcal{S}$. Let τ be a position in $\overline{E_e}$. Let e' be an element to be inserted into E_e ($e' \in \Sigma$). Let $\alpha : E_\alpha$ be a rule not in $\mathcal{S}$. The operations $ins(e, e', \tau, context, mode, times)$, $makeOpt(e, \tau)$, $ExtendCard(e, \tau)$ and $createCM(\alpha, E_\alpha)$ applied over $\mathcal{S}$ build a new schema $\mathcal{S}'$ such that $L(\mathcal{S}) \subseteq L(\mathcal{S}')$.* □

Proof (Sketch): A content model E_e can be updated as follows: (i) a new optional element can be inserted, (ii) a mandatory element can be transformed into an optional one, (iii) an element can have its cardinality augmented. By definition of regular expressions, all these updates performed on E_e, outputting E'_e, verify $L(E_e) \subset L(E'_e)$, thus $L(\mathcal{S}) \subseteq L(\mathcal{S}')$. Moreover, a new content model can be inserted associated to an element α. Here again, the documents valid with respect to S are still valid with respect to the new schema $\mathcal{S}'$, because either they do not contain α, or they contain α but its original content model is still a valid one (cf. Definition 3.4). □

High Level Primitives We can propose some high level primitives to express complex updates in a more compact way. Let E be the content model of element e.

1. Inserting a subexpression: $insSubExp(e, \beta, \tau, context, mode, times)$, where β is a regular expression. This operation inserts β into E relatively to τ or to its context (depending on the $context$ parameter), following the way expressed in $mode$ and with or without repetition (parameter $times$).

2. Making optional: $makeSubExpOpt(e, \beta)$, β being a subexpression of E.
3. Extending the cardinality: $ExtendSubExp(e, \beta)$, β being a subexpression of E.

These high level primitives, again inspired from [Mesiti et al. 2006], are also to be proposed via an intuitive interface to assist the user in expressing his/her needs. Instead of dealing with one node each time, these primitives imply to work with a set of nodes and edges (a subgraph), but they keep the same semantics as the atomic ones.

4. Conclusion

In this paper, we consider a framework in which a user can specify updates on XML schemas using an intuitive interface, and asks for these updates to be consistency-preserving (it is not necessary to revalidate the previous valid documents). We propose a method to perform the accurate update, such that the updated schema both respects all the specifications given by the user, and is a conservative extension of the original schema. This property is important, specially in a distributed environment where documents located in different sites have to respect a common schema $\mathcal{S}$. The update of $\mathcal{S}$ may make those documents invalid, and a distributed applications that uses $\mathcal{S}$ may no more work properly.

Our method is based on a graph-to-regular-expression reduction technique, which allows to identify the starred sub-expressions of a regular expression. Another way to find the starred sub-expressions and their behavior in E, *e.g.*, optionality, cardinality, etc., is to use the functions $First$, $Last$, and $Follow$ [Bruggeman-Klein 1993] that work directly on the regular expression E. Computing these functions is equivalent to build the Glushkov graph, the graph without orbits and the hierarchy of orbits corresponding to E.

The set of primitives proposed in this work is not complete. The missing primitives, *e.g.*, deletion or replacement of an element (or a sub-expression), are operations that inevitably make the XML database inconsistent [Guerrini et al. 2005]. Our primitives are to be used together with *non-conservative* primitives in a general framework such as the one presented in [Mesiti et al. 2006], that contains other tools for limiting revalidation and changes performed on existing documents when non-conservative updates are performed. Such a framework represents a solution to the increasing demand for tools specially designed for administrators not belonging to the computer science community, this demand been particularly strong in the domain of the Web and XML.

This work is part of a broader project that is exploring all faces of valid XML documents: (incremental) validation [Bouchou and Halfeld Ferrari 2003, Bouchou et al. 2003], constraint checking [Abrão et al. 2004], schema evolution triggered by updates in documents [Bouchou et al. 2004], correction of updates [Bouchou et al. 2006] and primitives for schema updates (this work). A prototype of each part of this project has been implemented in Java. We plan to integrate all these prototypes, while developing a complete and usable human-machine interface.

References

Abrão, M. A., Bouchou, B., Halfeld Ferrari, M., Laurent, D., and Musicante, M. (2004). Incremental constraint checking for XML documents. In *XSym*, volume 3186 of *Lecture Notes in Computer Science*, pages 112–127. Springer.

Al-Jadir, L. and El-Moukaddem, F. (2003). Once upon a time a DTD evolved into another DTD... In *OOIS*, volume 2817 of *Lecture Notes in Computer Science*. Springer.

Bouchou, B., Cheriat, A., Halfeld Ferrari, M., and Savary, A. (2006). XML Document Correction: Incremental Approach Activated by Schema Validation. In *IDEAS'06*.

Bouchou, B., Duarte, D., Halfeld Ferrari, M., and Laurent, D. (2003). Extending tree automata to model XML validation under element and attribute constraints. In *ICEIS*.

Bouchou, B., Duarte, D., Halfeld Ferrari, M., Laurent, D., and Musicante, M. A. (2004). Schema evolution for XML: A consistency-preserving approach. In *Mathematical Foundations of Computer Science*, number 3153 in LNCS, pages 876 – 888.

Bouchou, B. and Halfeld Ferrari, M. (2003). Updates and incremental validation of XML documents. In *The 9th DBPL*, number 2921 in LNCS.

Bruggeman-Klein, A. (1993). Regular expressions into finite automata. *Theoretical Computer Science*, 120:197–213.

Bruggeman-Klein, A. and Wood, D. (1992). Deterministic regular languages. In *STACS*.

Caron, P. and Ziadi, D. (2000). Characterization of Glushkov automata. *TCS: Theorical Computer Science*, 233:75–90.

Costello, R. and Schneider, J. C. (2000). Challenge of XML schemas - schema evolution. In *Proceedings of XML Europe*.

Guerrini, G., Mesiti, M., and Rossi, D. (2005). Impact of XML schema evolution on valid documents. In *WIDM*, pages 39–44. ACM.

L., D. and Chu, W. W. (2000). Comparative analysis of six XML schema languages. *SIGMOD Record*, 29(3):76–87.

Mesiti, M., Celle, R., Sorrenti, M. A., and Guerrini, G. (2006). X-Evolution: A system for XML schema evolution and document adaptation. In *EDBT*, pages 1143–1146.

Murata, M., Lee, D., Mani, M., and Kawaguchi, K. (2005). Taxonomy of XML schema language using formal language theory. *ACM Transactions on Internet Technology (TOIT)*, 5(4):660–704.

Papakonstantinou, Y. and Vianu, V. (2000). DTD inference for views of XML data. In *ACM Symposium on Principles of Database System*, pages 35–46.

Prashant, B. N. and Kumar, P. S. (2006). Managing XML data with evolving schema. In *COMAD*, pages 168–175. Computer Society of India.

Raghavachari, M. and Shmueli, O. (2004). Efficient schema-based revalidation of XML. In *EDBT*, LNCS, pages 639–657. Springer.

Roddick, J., Al-Jadir, L., Bertossi, L., Dumas, M., Estrella, F., Gregersen, H., Hornsby, K., Lufter, J., Mandreoli, F., Männistö, T., Mayol, E., and Wedemeijer, L. (2000). Evolution and change in data management - issues and directions. *SIGMOD Record*, 29(1):21–25.

Su, H., Kuno, H., and Rundensteiner, E. A. (2001). Automating the transformation of XML documents. In *3rd WIDM*. ACM.