

# A Structural Adviser for the XML Document Authoring

Boris Chidlovskii  
Xerox Research Centre Europe, France  
6, chemin de Maupertuis, F-38240 Meylan, chidlovskii@xrce.xerox.com

## ABSTRACT

Since the XML format became a *de facto* standard for structured documents, the IT research and industry have developed a number of XML editors to help users produce structured documents in XML format. However, the manual generation of structured documents in XML format remains a tedious and time-consuming process because of the excessive verbosity and length of XML code. In this paper, we design a structural adviser for the XML document authoring. The adviser intervenes at any step of the authoring process to suggest one tag or entire tree-like pattern the user is *most likely* to use next. Adviser suggestions are based on finding analogies between the currently edited fragment and *sample data* being either previously generated documents in the collection or the history of the current document authoring. The adviser is beneficial in cases when no schema is provided for XML documents, or schema associated with the document is too general and sample data contain specific patterns not captured in the schema. We design the adviser architecture and develop a method for efficient indexing and retrieval of optimal suggestions at any step of the document authoring.

## Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation—*Markup languages*; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

## General Terms

Algorithms, Measurement, Documentation, Performance

## Keywords

XML markup, structural pattern, suggestion, data mining

## 1. INTRODUCTION

XML is a standard format for information exchange in IT applications and systems. The uniform data format for semantic tagging of data provided by XML and document models in the form of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*DocEng '03*, November 20–22, 2003, Grenoble, France.  
Copyright 2003 ACM 1-58113-724-9/03/0011 ...\$5.00.

DTDs or XML Schemas simplify data exchange between heterogeneous information systems or different components of one system. However, the amount of documents available/generated in XML format remains fairly low as compared to documents in other formats. First, there exists a nontrivial problem of converting documents from other formats into XML. Second, creating new XML documents appears to be a time-consuming process, because of a particular verbosity and lengthiness of XML documents. The need to permanently interleave document content (textual data) with semantic tags and attributes in XML documents makes the generation process tedious and error-prone.

A number of commercial XML editors (XML Spy, Xeena, XMetaL, FrameMaker, ElfData, Morphone, etc.) and public ones<sup>1</sup> help the XML document authors partially reduce the document authoring overhead by offering an advanced graphic interface with menu-based selection of elements/attributes and a possibility to align the document generation with a corresponding DTD or XML Schema by validating entire documents or their fragments. Though DTDs and XML Schema serve well for the document validation, they provide a limited help during the document authoring.

Two main reasons are the following. First, in majority of cases, schemas are designed by humans *before* any valid documents are created. Schemas represent information models for business scenarios and processes in a certain domain, and often reflect an agreement of various players in the domain. As result many schemas are too general, they target the validation only, but allow much more degree of generality than the real documents do expose.

Second, a schema can help find out valid tags during the document authoring, suggesting more complex patterns like *tree-like patterns*, is not immediate with schema mechanisms. In DTDs, element definitions are *extended regular expressions* describing (infinite) sets of possible element contents, while the document authoring can be seen as a sequential instantiation of these element definitions. In XML Schemas, complex types define sets of tree patterns, they however are mostly used for the convenience of encapsulation and reuse and not assumed to reflect more representative or frequent patterns in data.

In this paper, we design a *structural adviser for the XML document authoring* that assists the author in the creation of document structure, thus leaving to the user the conventional work of content generation. Adviser's suggestions are based on mining available data, given either by the history of current document authoring or by existing documents with a common schema. We propose a method that mines the sample data in order to suggest a single tag or a tree pattern the author is most likely to use next. We design the adviser architecture and discuss requirements an adviser

<sup>1</sup>See <http://www.oasis-open.org/cover/publicSW.html> for details.

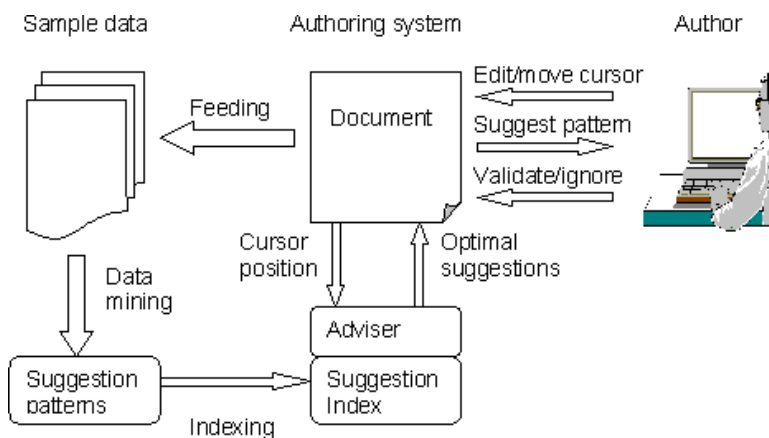


Figure 1: Generic architecture of an adviser component for the document authoring.

should satisfy. The method to analyze sample data, determine suggestion candidates and select an optimal suggestion at any cursor position in a current document. The adviser appears to be a *real-time data-intensive application*, as it requires focused techniques for data mining and indexing suggestions in order to support a fast retrieval of optimal candidates in an interactive authoring environment.

The remainder of the paper is organized as follows. Section 2 introduces the architecture of structural adviser for XML authoring and analyzes the core requirements an adviser should satisfy. It gives some examples of how the adviser works. Section 3 provides a formal definition of the structural advising problem and Section 4 introduces the aggregate similarity function for selecting optimal candidates. Section 5 introduces an efficient data structure for the basic case of indexing the suggestions. Section 6 reviews the prior art in mining and indexing tree-like data. Section 7 reports on results of adviser experiments we have run with various collections of real and synthetic XML data; these results demonstrate its remarkable effectiveness: the adviser reduces the the cost of a document structure generation in 4 to 20 times. The discussion of open issues completes Section 7 and Section 8 concludes the paper.

## 2. ARCHITECTURE AND REQUIREMENTS

The architecture of the structural adviser is designed in Figure 1. Sample data is mined to identify suggestion patterns and to index them; the adviser retrieves and offers one or several optimal patterns in a context-dependent manner, for a given cursor position in a currently authored document. User can accept or ignore the suggestions; in turn, any new structural changes in the current document might feed the sample data and trigger the refinement of suggestion patterns.

The adviser should be able to intervene at any position of the document structure, that is, each time the author opens/closes a tag, change a cursor position, etc. At any given cursor position in the document, the user cannot add arbitrary tags, but only tags allowed by the referred schema. For example, the element definition  $a := (b|c)^*$  allows either tag (element)  $b$  or  $c$  at any position within element  $a$ , without prioritizing any of the two.

In the general case, schemas associated with XML documents might be *tight* or *loose*; tight schemas define a highly rigid and regular structure, thus allowing mostly one tag at most positions in a document; such schemas do occur in data-centric XML col-

<sup>2</sup>In the DTD syntax, it is denoted as  $\langle !ELEMENT a (b|c)^* \rangle$ .

lections. However, tight schemas are rather an exception, a much more frequent case is that of loose schemas; their design is often driven by certain agreements and compromises (partners, domain, etc.); such schemas offer multiple possible tags at most positions in a document.

Our structural adviser is built around a set of suggestion patterns; such patterns are extracted by a data mining component from available *sample data*. Data mining can be done *on-line* or *off-line*. Off-line data mining takes place when the sample data is a collection of documents provided in advance or previously generated by users. The suggestion patterns extracted and indexed off-line remain unchanged during all the session of a new document authoring, until the sample collection is extended with the new document(s) and the system can refine suggestion patterns and their probabilities from the updated collection.

Alternatively, the patterns can be identified *on-line*, during the process of a current document authoring. Sample data is initially empty and grows as long as the user edits the document. Suggestion patterns and overall their probabilities are determined incrementally, each elementary authoring action can (immediately or with some delay) update the suggestion index, since any addition of a new tag or removal of an old one changes the frequencies of tags/patterns and therefore can alter possible suggestions by the adviser.

The XML structural adviser is a real-time data-intensive application. On one side, it should mine sample data in order to identify optimal candidates for suggesting. On the second hand, it should be pro-active in an interactive authoring environment and moreover support *valid* suggestions at any cursor position in a document. It should organize candidates in such an efficient way that for any change in the document of even cursor position, the retrieval of the optimal candidate should be as imminent.

EXAMPLE 1. Assume that one or multiple XML documents with an associated schema are provided for the mining component that has identified suggestion patterns for authoring new documents. Assume the user creates a new document with the same schema, and edits, say, an element  $a$ . Below we consider two cases, where each case includes how element  $a$  is constrained by the schema, patterns for  $a$  are in sample data and their occurrences, and how the adviser can suggest the most relevant tag or pattern. In Table 1 that gives detail of each case, pattern  $a(b, b)$  denotes a node  $a$  with two subtrees both labeled  $b$ ;  $\uparrow$  denotes a current cursor position; formal definitions will be given in the following section.

Case	DTD constraints	Sample data		Cursor	DTD allows	Suggested tag	Suggested pattern
		Patterns	Occurrences				
1.1	$a := (b c)^*$	$a(b, b)$	2	$a(\uparrow)$	$b$ or $c$	$b$	$(b, b)$
1.2		$a(b, b, c)$	5	$a(c \uparrow)$	$b$ or $c$	$c$	$(c)$
		$a(c, c)$	3				
2	$d := (a+)$ $e := (a+)$ $a := (b c)^+$	$d(a(b, b))$	2	$e(a(\uparrow))$	$b$ or $c$	$c$	$(c, c)$
		$d(a(b, b, c))$	5				
		$e(a(c, c))$	3				

**Table 1: Two cases of the structured advisor at work.**

*Case 1. The schema constrains the element  $a$  as  $a := (b|c)^*$ , thus allowing any  $a$  to contain any sequence of  $b$  or  $c$ . For a new open element  $a$ , the adviser suggests tag  $b$  as the most likely first tag in  $a$ ; such a decision the adviser makes by observing sample data where  $a$  starts with  $b$  in 7 cases of 10. The adviser equally suggests pattern  $(b, b)$ . If element  $a$  already contains tag  $c$  (sub-case 1.2), the adviser suggest to add tag  $c$ , that is intuitively coherent with pattern occurrences.*

*Case 2. Analysis of sample data may go beyond simple mutual tag occurrences. Assume that the schema, beyond  $a := (b|c)^*$ , has constraints  $d := (a+)$  and  $e := (a+)$  and sample data are such as in Table 1. Then, the adviser can find out that  $a$  starts with  $b$  when  $a$ 's father is  $d$ , while  $c$  is the first tag in  $a$  when  $a$  has  $e$  as father. Thus the adviser may mine the context of tag  $a$ , that is, which tag precedes it. In case 2 in Table 1 tag  $a$  is preceded by tag  $e$ , thus adviser suggests tag  $c$  and pattern  $(c, c)$  as the most relevant ones.*

At any step of the document authoring, the adviser should supply most *relevant* suggestions, these suggestions and their probabilities being evaluated from the sample data. The success of the adviser is measured by the ratio of good advises and how do they reduce the document generation overload. The induction of good suggestions requires a focused analysis of structural patterns in sample data.

Like in data mining, adviser mines sample data to find out tree patterns meeting criteria of valid candidate. On the other hand, like in information retrieval, adviser has to rank patterns according to a certain relevance function, in order to suggest one of few most relevant patterns. Moreover, adviser should cope with its own problems. First, adviser should operate in the contextual manner, where the context is given by a cursor position and its surroundings. Second, candidates for suggestions differ in both size and frequency, therefore the adviser should be capable to uniformly compare such different candidates in order to select the best one.

In total, the XML structural adviser is based on methods of data mining, tree indexing and relevance estimation. Any efficient and effective solution for the structural adviser should meet following major requirements:

- Adviser's suggestions should be contextual and address a current cursor position;
- Defining the space of suggestion candidates should be also contextual and be driven by tags surrounding the cursor;
- Selection of optimal suggestions should be based on the gain measure and similarity with patterns in sample data;
- Suggestion candidates should be organized in the form of an index, in order to support their fast retrieval.

In the following sections, we develop a method for determining optimal patterns from sample data and a index structure for pattern storage and retrieval.

### 3. DATA MINING FOR SUGGESTING

At any step of the document authoring, the adviser considers a set of candidates for both one-tag and pattern suggestions and detects the optimal ones for either case. The *optimal candidate* is one that is the most relevant (but not necessarily the most probable) for the next document edition step; it maximizes a certain *gain function* based on similarity with patterns found in sample data. We note that suggesting a tree pattern is more difficult than suggesting one tag, since the pattern suggestion should cope with the problem of selection among candidates of different size. The problem comes from the fact that small-size patterns are always more frequent than large-size ones. On the other hand, proposing a large pattern may be more beneficial because, if accepted, a large pattern further reduces the edition overhead that a small one. The method we develop is from the best suggestions represent an *optimal trade-off* between candidates of different sizes and their frequencies. In the following, we address mainly the general case of finding optimal patterns, while the one-tag suggestions are considered as a special case when the pattern size is limited to 1.

#### 3.1 Problem definition

We consider an XML document as a tree where inner nodes determine the structure of document, and the leaf nodes and the tag attributes provide the document content. We follow [8] in abstracting XML documents as the class of *unranked labeled rooted trees*. Tree  $T$  is defined over an alphabet  $\Sigma$  of tag names. The set of trees, denoted by  $S_\Sigma$ , is inductively defined as follows:

1. every  $\sigma \in \Sigma$  is a tree leaf<sup>3</sup>,
2. if  $\sigma \in \Sigma$  and  $t_1, t_2, \dots, t_n \in S_\Sigma, n \geq 1$ , then  $\sigma(t_1, t_2, \dots, t_n)$  is a tree in  $S_\Sigma$ .

There is no a priori bound on the number of children of a node in a tree; such tree are therefore unranked. In a node  $\sigma(t_1, t_2, \dots, t_n)$ ,  $\sigma$  is a root and  $t_1, t_2, \dots, t_n$  are subtrees. A *forest* is any subset of subtrees  $(t_{i_1}, \dots, t_{i_m}), 1 \leq i_j \leq n$ . For any rooted subtree  $t$  in tree  $T \in S_\Sigma$ , the *path* from the tree root to the subtree is denoted  $p(t)$ , the *depth* of  $t$  is denoted  $d(t)$  (leaf nodes have depth 0); the *size* of  $t$  is denoted  $|t|$  and equals to the number of nodes in the subtree.

The set of trees  $S_\Sigma$  can be constrained by a *schema*  $D$  that is defined as a set of constraints  $C(\sigma) \in D$  for some or all  $\sigma \in \Sigma$ . Node  $\sigma(t_1, t_2, \dots, t_n)$  in a tree  $T \in S_\Sigma$  *conforms* to schema  $D$  if the forest  $(t_1, t_2, \dots, t_n)$  satisfies the formula  $C(\sigma)$  in  $D$ . Tree

<sup>3</sup>Leaf  $\sigma$  will be equally denoted  $\sigma()$ , a node having zero subtrees.

$T$  conforms to schema  $D$  if all its nodes do so. The class of trees  $S_\Sigma$  conforming to all schema constraints in  $D$  is denoted  $S_\Sigma^D \subseteq S_\Sigma$ . In the following, we will consider the case when schema  $D$  is given by XML DTD specification according to which  $D$  contains a constraint formula  $C(\sigma)$  for each  $\sigma \in \Sigma$ , and each constraint  $C(\sigma)$  is defined as an extended regular expression over  $\Sigma' = \Sigma \cup \{PCDATA\}$  [13]. In this case, node  $\sigma(t_1, t_2, \dots, t_n)$  in a tree  $T$  confirms the schema if  $t_1, t_2, \dots, t_n$  is a string in the extended regular language defined by  $C(\sigma)$ .

### 3.1.1 Prefix trees

The notion of *prefix tree* plays a core role in our approach to the suggestion of tree patterns. It is defined as follows:

1. tree  $\sigma(t_1, t_2, \dots, t_m)$  is a prefix of tree  $\sigma(t_1, t_2, \dots, t_n)$ , if  $0 \leq m \leq n$ , where case  $m = 0$  refers to a leaf node  $\sigma()$ ,
2. tree  $\sigma(t'_1, t'_2, \dots, t'_m)$  is a prefix of tree  $\sigma(t_1, t_2, \dots, t_n)$ , if  $m \leq n$  and  $t'_i$  is a prefix tree of  $t_i, i = 1, \dots, m$ .

Informally, tree  $T' \in S_\Sigma$  is a *prefix* of tree  $T \in S_\Sigma$  if  $T$  can be obtained from  $T'$  by appending zero or more subtrees to some of nodes in  $T'$ . Note any tree  $T$  is a prefix of itself. The set of appended subtrees is a forest. A forest  $f$  is a prefix of forest  $f'$  if there exists  $\sigma \in \Sigma$  such that tree  $\sigma(f)$  is a prefix of tree  $\sigma(f')$ .

**EXAMPLE 2.** Let alphabet  $\Sigma$  be  $\{a, b, c, d, e\}$ . Consider three trees over  $\Sigma, T_1 = d(a(b, b), a(b, b, c)), T_2 = e(a(e, d), c, c)$  and  $T_3 = d(a(b, b), a)$ , see Figure 2. Let schema  $D$  over  $\Sigma' = \{a, b, c, d, e, PCDATA\}$  constrains  $b$  and  $c$  to be leaves ( $b := PCDATA, c := PCDATA$ ) and three constrains internal nodes  $a, c$  and  $d$  as  $a := (b|c)^+, c := d := a^*$  (see Example 1). The schema definition  $D$  constrains a class  $S_\Sigma^D$  of trees that conform to  $D$ . In Figure 2, only tree  $T_1$  is in  $S_\Sigma^D$ , for  $T_2$  violates constraints  $C(a)$  and  $C(c)$ , and  $T_3$  violates  $C(a)$ . Finally, tree  $T_3$  is a prefix of tree  $T_1$ .

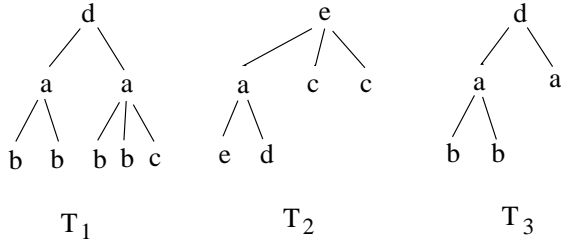


Figure 2: Three example trees.

## 3.2 Authoring process

When the user is authoring a document  $T$ , the authoring process is considered as a sequence of elementary actions on the document structure,  $T^0, T^1 = T^0 + \delta_0, \dots, T^{j+1} = T^j + \delta_j, \dots, T^N$ , where an elementary action  $\delta_j$  is an addition/removal of a forest or change of cursor position. The final version  $T^N$  conforms to the document schema  $D$  and therefore  $T^N \in S_\Sigma^D$ , but few or even none intermediate versions  $T^j, j < N$ , may conform to  $D$ .

### 3.2.1 Cursor position and context

The adviser supplies suggestions at any cursor position in a tree  $T$ , when the cursor (denoted  $\uparrow$ ) points to a position between two tags in a tree and where a new tag or a subtree can be inserted. Formally, a cursor position is defined by the *absolute location position*

$r$  of the tag that immediately precedes the cursor. We represent  $r$  as a tuple  $r = (\sigma_r, p(\sigma_r), f_p)$ , where  $\sigma_r \in \Sigma$  is the *context node*<sup>4</sup> of  $r$ , that is the closest node containing the cursor;  $p(\sigma_r)$  is the absolute location path of  $\sigma_r$  in the tree, and forest  $f_p$  is  $\sigma_r$ 's content preceding the cursor. As an example, in a tree  $T_r = d(a(b, b \uparrow), a)$ , the cursor position is  $r = (a, /d/a[1], (b, b))$ . Positioning the cursor in a newly created tag  $a$  is denoted as  $a(\uparrow)$ .

For a cursor position  $r$  in  $T$ , the schema  $D$  allows a set  $D(T_r) \subseteq \Sigma$  of tags. This set can be determined as follows. From the schema constraint  $C(\sigma_r)$ , we can build a finite state automaton  $A$  for the extended regular expression defined by  $C(\sigma_r)$ .  $D(T_r)$  is not empty if processing the prefix  $f_p$  with automaton  $A$  matches any (final or not) state  $q$  in  $A$ . In this case, the set  $D(T_r)$  is given by labels of outgoing transitions from state  $q$  [13]. The method for  $D(T_r)$  can be extended to enumerate valid multi-tag patterns, however the construction of such a pattern set is beyond the scope of this paper.

For the structural adviser, we start by considering valid candidates for cursor position  $r$  in  $T$ . Forest  $f$  is a *valid candidate* for  $T_r$  if tree  $T$  after adding  $f$  at position  $r$  does satisfy the constraint  $C(\sigma_r)$  of  $D$ .  $VC(T_r)$  will denote the set of all valid candidates for cursor position  $r$  in  $T$ .

In our proposal, the adviser takes into consideration only certain *limited context* surrounding the cursor position  $r$  in  $T$ . If  $x(T_r)$  denotes a certain context for  $T_r$ , the selection of candidates for advising is defined by this context only, that is,

$$VC(T_r) = VC(x(T_r)).$$

In Example 1 in Section 2, two variants of the context definition have been used. Case 1 used so-called *minimal context*, defined as  $x_m(T_r) = (\sigma_r, f_p, f_s)$ ; it takes into account the context node  $\sigma_r$  and the split of  $\sigma_r$ 's subtrees by the cursor into two parts, called *prefix and suffix forests*,  $f_p$  and  $f_s$ , respectively. In case 2, we used the *father context* that extends the minimal context with the father node; it is defined as  $x_f(T_r) = (\sigma_r, \sigma_f, f_p, f_s)$ , where  $\sigma_f \in \Sigma$  is the father of  $\sigma_r$  in  $T$ .

## 3.3 Gain function

Now we introduce the concept of gain function  $G(T_r, f)$  that estimates the relevance of a candidate  $f \in VC(T_r)$  in the case if  $f$  is accepted by the author. Among all candidates in  $VC(T_r)$ , the adviser will suggest one (or few) with the maximal gain. This optimal suggestion  $f_{opt}$  is therefore given by

$$\begin{aligned} f_{opt} &= \operatorname{argmax}_{f \in VC(T_r)} (G(T_r, f)) \\ &= \operatorname{argmax}_{f \in VC(x(T_r))} (G(T_r, f)). \end{aligned} \quad (1)$$

In Example 1, we have seen that taking a wider context may considerably change the evaluation of candidates and optimal suggestions. Beyond the context, the adviser can consider for suggesting sub-trees of different size and depth. On the other hand, taking too large context and deep sub-tree candidates faces the problem of managing an exponentially increasing number of candidates and complicates considerably their management and efficient retrieval of the optimal candidate. Therefore, a good adviser should find out a good *trade-off* between considered context and complexity of candidate management. In the following section, we develop a method driven by the two main parameters, the context  $x(T_r)$  and the maximal suggestion depth  $d$ . For the sake of simplicity, we will mostly use the minimal context  $x_m$ , making when needed, remarks for the father context  $x_f$ .

<sup>4</sup>In XPath terminology, see <http://www.w3c.org/TR/xpath20/> for more details.

To select an optimal suggestion at a given cursor position  $r$ , the adviser should (1) constrain the set  $VC(T_r)$  of suggestion candidates, and (2) measure the *similarity* between any candidate  $f$  in  $VC(T_r)$  and tree patterns occurred in the sample data in the same context  $x(T_r)$  and (3) pick up such  $f$  that maximizes the gain function that minimizes  $G(T_r, f)$ . To develop an efficient solution for the structural adviser, we believe that the similarity and gain functions should satisfy three following requirements:

1. *Simplicity*: The similarity and gain functions should be easily computed from the sample data.
2. *Effectiveness*: A similarity measure between a candidate  $f$  and the pattern set should provide a good trade-off between size and frequency of candidates.
3. *Incrementality*: In the off-line case, the similarity and gain values should not be recomputed at each new step of document authoring. Changing context (due to an advance in the document authoring) may alter or reduce the candidate set, but it should not change their similarity values. In the on-line case, the similarity and gain may be recomputed only for patterns relevant to the last changes in the document.

In the following section, we develop one formulation for similarity and gain functions that address these three requirements.

## 4. CONTEXT AND SUGGESTIONS

In the minimal context  $x_m = (\sigma_r, f_p, f_s)$ , we distinguish between its *core* given by  $\sigma_r$ <sup>5</sup> and *content* is given by  $f_p$  and  $f_s$  forests. We treat the suggestion selection problem in two phases. We first analyze a simpler case of empty content  $f_p = f_s = ()$  (when the cursor is positioned inside a new tag) and show how to select optimal suggestions. Then, in the next subsection, we will show how the solution for the empty content can be extended to the non-empty content.

### 4.1 Suggestions for empty content

Let  $S$  denote the sample data being either one or multiple XML documents that conform to schema  $D$ . For a context definition  $x$ ,  $X$  denotes the set of different core values of  $x$  in  $S$ . For the minimal context  $x_m$ , we have  $X_m = \{\sigma_r\} \subseteq \Sigma^6$ . Assume we consider patterns of the depth  $d$  for a new opened tag  $\sigma_r \in X_m$ . Then,  $S(\sigma_r) = \{f_i\}$  will denote the set of forests in sample data, where each forest  $f_i$  is a content of a sub-tree rooted at  $\sigma_r$  of depth  $d$ . All  $f_i$  are given with their probability (normalized frequency)  $pr_i$ , that is,  $\sum_i pr_i = 1$ .

Now we define a similarity function that satisfies the requirements mentioned in the previous section. The candidate set for the empty content case is  $VC(\sigma_r) = VC(\sigma_r, (), ())$ . We build the candidate set  $VC(\sigma_r)$  as the set of all forests in  $S(\sigma_r)$  extended with all their valid prefixes:

$$VC(\sigma_r) = \{f | f \text{ is a prefix of } f_i \in S(\sigma_r) \text{ and } f \in C(\sigma_r)\}. \quad (2)$$

The similarity function  $sim$  between a candidate  $f \in VC(\sigma_r)$  and a tree pattern  $f_i \in S(\sigma_r)$  is defined as follows:

$$sim(f, f_i) = \begin{cases} |f|/|f_i|, & \text{if } f \text{ is a prefix of } f_i, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

<sup>5</sup>For the father context  $x_f = (\sigma_r, \sigma_f, f_p, f_s)$ , the core is given by a pair  $(\sigma_r, \sigma_f)$ .

<sup>6</sup>For the father context  $x_f$ ,  $X_f$  is given by all different pairs  $(\sigma_r, \sigma_f)$  in sample data and therefore,  $X_f \subseteq \Sigma^2$ .

Note that  $sim(f, f_i)$  takes values between 0 and 1; moreover  $sim(f, f_i) = 1$  iff  $f = f_i$ .

The gain function  $G(T_r, f)$  for candidate  $f$  for the empty context is defined as the *aggregate similarity function* over all patterns in  $S(\sigma_r)$ :

$$G(T_r, f) = G(\sigma_r, f) = \sum_{f_i \in S(\sigma_r)} sim(f, f_i) \cdot pr_i. \quad (4)$$

The optimal suggestion  $f_{opt}$  is then given by a such candidate  $f \in VC(\sigma_r)$  that maximizes the gain function  $G$ :

$$f_{opt} = \operatorname{argmax}_{f \in VC(\sigma_r)} (G(\sigma_r, f)).$$

**EXAMPLE 3.** Take again case 1.1 in Example 1 and three patterns of depth  $d=1$  for tag  $a$  in sample data,  $S(a) = \{(b, b), (b, b, c), (c, c)\}$  with respective probabilities 0.2, 0.5 and 0.3 (see Table 1 for the pattern occurrences). The candidate set for an opened tag  $a$  is given by set  $S(a)$  extended with their prefixes  $VC(a) = \{(b), (c), (b, b), (b, b, c), (c, c)\}$ . Note that all candidates in  $VC(a)$  conform to the constraint  $C(a) = (b|c)^+$ . For the first candidate  $(b)$ , we calculate its similarity with patterns in  $S(a)$ :

$$\begin{aligned} sim((b), (b, b)) &= 0.5, \\ sim((b), (b, b, c)) &= 0.33, \\ sim((b), (c, c)) &= 0. \end{aligned}$$

Thus we obtain the aggregate similarity value for  $(b)$ :  $G((b), S(a)) = 0.5 * 0.2 + 0.33 * 0.5 = 0.26$ . In a similar way, we evaluate the gain for other candidates in  $VC(a)$ :

$$\begin{aligned} G((b, b), S(a)) &= 0.53, \\ G((b, b, c), S(a)) &= 0.5, \\ G((c), S(a)) &= 0.15, \\ G((c, c), S(a)) &= 0.3. \end{aligned}$$

Therefore, the optimal suggestion  $f_{opt}$  that maximizes the gain for the opened tag  $a$  is  $(b, b)$ .

For one-tag suggestions, we constrain the candidate set  $VC(\sigma_r)$  to only one-tag pattern,  $VC_1(\sigma_r) = \{f \in VC(\sigma_r), |f| = 1\}$  and determine the optimal candidate in the same manner. In the example above,  $C_1$  contains two one-tag candidates,  $C_1(a) = \{(b), (c)\}$ , and  $b$  is the optimal one-tag suggestion.

### 4.2 Suggestions for non-empty content

Example 3 above explains the method of finding the optimal suggestions for the empty content case. Let us now consider the case of non-empty content, when either  $f_p$  or  $f_s$  is not  $()$ . In Example 3, assume the user has accepted the suggested pattern  $(b, b)$  for the new tag  $a$  and place cursor after it:  $a(b, b \uparrow)$ . What the adviser should propose next? The non-empty prefix  $f_p = (b, b)$  will constrain the candidate set, but it will keep the evaluation of optimal candidates unchanged. The candidate set for prefix  $f_p$ ,  $VC(\sigma_r, f_p)$  is ordinarily defined as a subset of  $VC(\sigma_r)$  as follows:

$$VC(\sigma_r, f_p) = \{f \in VC(\sigma_r) | f_p \text{ is a prefix of } f\}. \quad (5)$$

In a similar way,  $VC_1(\sigma_r, f_p) = \{f \in VC(\sigma_r) | f_p \text{ is a prefix of } f, |f| = |f_p| + 1\}$  is a set of one tag candidates. For our example, we have  $f_p = (b, b)$ ,  $VC(\sigma_r, f_p) = VC(a, (b, b)) = \{(b, b), (b, b, c)\}$  and  $VC_1(a, (b, b)) = \{(b, b, c)\}$ . The reduction of the candidate set leads to the re-weighting of probabilities  $pr_i$  of patterns remained in  $CV(\sigma_r, f_p)$ . Although the re-weighting of pattern probabilities will change the absolute gain values, it will not change their relative order. This observation can be generalized to any content given the prefix and suffix forests  $f_p$  and  $f_s$ , as follows:

**PROPOSITION 1.** For any core value  $\sigma_r \in X_m$  and content  $(f_p, f_s)$ , if two candidates  $f_1$  and  $f_2$  are both in  $VC(\sigma_r)$  and  $VC(\sigma_r, f_p, f_s)$ , then  $G((\sigma_r, f_p, f_s), f_1) < G((\sigma_r, f_p, f_s), f_2)$  only if  $G(\sigma_r, f_1) < G(\sigma_r, f_2)$ .

Therefore, we can reuse the evaluation of optimal suggestions done for the empty content  $VC(\sigma_r)$ . Since  $(b, b)$  is the optimal pattern candidate in  $VC(a, (b, b))$ , the adviser will suggest to close tag  $a$  (empty pattern). Similarly, tag  $c$  is the (only) one-tag suggestion.

Proposition 1 assumes the minimal context  $x_m$ , but it can be directly extended to the father context  $x_f$ . Moreover, suggestions for non-empty content cases with  $x_m$  permits us to organize suggestions for both empty and non-empty content cases with  $x_f$ . Indeed, in case 2 of Example 1, we could have considered the father of opening tag  $a$  in the same way we have considered the prefix  $(b, b)$  in Example 3. We build the sample pattern set  $S^{+1}(a)$  for  $x_f$  starting with the father node  $\sigma_f$  in the document, that is,  $S^{+1}(a) = \{d(a(b, b)), d(a(b, b, c)), e(a(c, c))\}$ . Once we have extended the context for element  $a$  with father tags  $d$  and  $e$ , we can proceed with the construction of the candidate set  $CV^{+1}(\sigma_r, \sigma_f)$  and determination of optimal suggestions for each content as before. For the case 2, we obtain  $T_r = (e(a(\uparrow)))$  and the candidate set  $CV^{+1}(\sigma_r, \sigma_f) = CV^{+1}(a, e) = \{(c), (c, c)\}$ . Gain values for candidates  $(c)$  and  $(c, c)$  are 0.15 and 0.3 and, therefore  $(c, c)$  is the optimal suggestion for  $T_r = (e(a(\uparrow)))$ .

The father context  $x_f$  extend the minimal context  $x_m$  with one contextual node, but can be generalized to the context of any depth, or eventually to the full path from the tree root to the context node  $\sigma_r$ .  $k$ -context of  $\sigma_r \in \Sigma$  is the sequence of ancestors of  $\sigma_r$  in a tree  $T$ ,  $(\sigma_1, \sigma_2, \dots, \sigma_k)$ , where element  $\sigma_i$  is an immediate ancestor of  $\sigma_{i+1}$  and  $\sigma_k$  is the immediate ancestor of  $\sigma_r$ .  $k$ -context pattern set  $S^{+k}(\sigma_r)$  in sample data  $S$  consists of all patterns rooted at  $\sigma_r$ , where each pattern come with the leading  $k$ -context of  $\sigma_r$ . Once the  $k$ -context pattern set  $S^{+k}(\sigma_r)$  is built, the candidate set  $CV^{+k}(\sigma_r)$  and optimal suggestions are determined as described in previous sections. When the adviser should suggest a pattern for an opening tag  $\sigma_r$ , it applies the  $k$ -context of  $\sigma_r$  as a prefix, to identify the optimal candidate.

## 5. SUGGESTION INDEX

The effective work of the structural adviser assumes that all candidates are quickly identified and retrieved, that is, for any cursor position  $r$  in tree  $T$ , the adviser can promptly identify the candidate set  $CV(T_r)$  and the optimal candidate  $f_{opt}$  in  $CV(T_r)$ . Here, we propose an *efficient data structure for storing and retrieval* of suggestion candidates. For the minimal context  $x_m$  and the sample pattern set  $S(\sigma_r)$ ,  $\sigma_r \in X_m$ , we represent the candidate set  $VC(\sigma_r)$  (with associated gain function values) in the form of an extended *prefix automaton PA*. This automaton accepts strings over  $\Sigma^*$ , has states and transitions as usual. The automaton has no cycles and each state  $q$  in  $PA$  corresponds to a candidate  $f \in VC(\sigma_r)$ ; the state  $q$  for  $f$  is labeled with the gain value  $G(\sigma_r, f)$ ; final states in  $PA$  correspond to patterns in  $S(\sigma_r)$ . Beyond the usual transitions, each state  $q$  is extended with a transition to the state  $q'$  for candidate  $f' \in VC(\sigma_r)$ , such that  $f'$  is the optimal pattern provided that  $f'$  is the prefix,  $f' = f_{opt}$  for  $VC(\sigma_r, f_p)$ .

Figures 3 and 4 show the prefix automata for both cases 1 and 2 in Example 1; usual and extended transitions are indicated with solid and dotted arcs; final states in  $PA$  are double-circled. Since all optimal suggestions are again states in  $PA$ , an optimal suggestion for state  $q = f$  is shown as a transition (a dotted arc) toward the corresponding state.

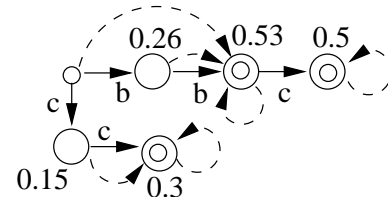


Figure 3: Extended prefix tree for case 1 in Example 1.

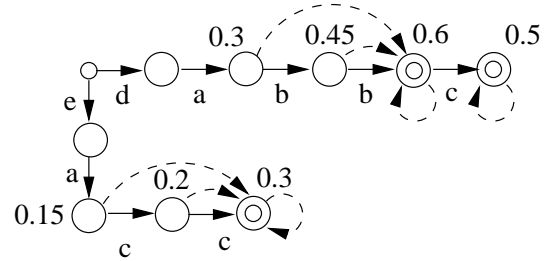


Figure 4: Prefix tree automaton for case 2 in Example 1.

Finding the optimal candidate for prefix  $f_p$  is as follows (the empty content case, the prefix  $f_p$  is  $()$ ). The prefix  $f_p$  corresponds to a state  $q$  in  $PA$  if only  $f_p \in VC(\sigma_r)$ . The candidate set for the prefix  $f_p$ ,  $VC(\sigma_r, f_p)$  is the set of states reachable from state  $q$  for  $f_p$  and the optimal candidate for the prefix  $f_p$  is found by following the dotted arc from the state  $q$ . For example, the initial state ( $f_p = ()$ ) of  $PA$  in Figure 3 refers to the state  $q' = (b, b)$  as the optimal suggestion; the state  $q$  for  $f_p = (b, b)$  refers to itself, that means null or “close-this-tag” suggestion.

In total, the structural adviser maintains a suggestion index that represents a set of extended tree automata. On the case of minimal context  $x_m$ , the index contains one automaton for each core value in set  $X_m$ . In the case of father context  $x_f$ , one automaton is provided for one  $\sigma_r$ , but possible for multiple pairs  $(\sigma_r, \sigma_f) \in X_f$  (see Figure 4).

## 6. PRIOR ART

Interactive editors for the document preparation have evolved from Rita [4] and Grif [6] edition systems, that used predefined document grammars to provide the context information and to guide the authoring process, to the recent editors and validation systems for XML documents, like Microsoft XML Notepad [9], XML-Spy [12], Corel XMetaL [1], IBM Xena [3, 11] and others. The editors provide an interactive interface for the manual creation, editing and browsing of XML data. The interfaces are often coupled with DTD/XML Schema grammars and content views in order to validate data against DTDs or XML Schema schema definitions and to facilitate the creation of XML documents.

Finding patterns in tree-like data is a core problem in various domains, like bioinformatics, Web mining, semi-structured data, etc. In the Web mining, the main interest is in the efficient enumeration of frequent trees in a data forest, where a frequent tree is a tree occurring at least *minsup* times. Enumerating all frequent patterns combines methods of efficient data mining [2, 5] and the tree pattern matching. [14] has recently presented TreeMiner, a novel time- and space-efficient algorithm for discovering all frequent subtrees in a forest. In the semi-structured data, methods of the extraction

schema also perform the statistical analysis of sample data in order to find out the most representative patterns [7, 10].

The structured adviser does also mine XML data and indexes sub-trees. Unlike the problems of mining frequent sub-trees mining or finding the representative tree patterns, the adviser first filters all tree patterns by their “applicability” at a given cursor position and, second, it ranks the selected patterns by their gain value, in order to suggest the best one.

## 7. MEASUREMENT AND EXPERIMENTS

We have run two series of experiments in order to validate the method of structural advising developed in Sections 2-5. For experiments, we have used two sources of natural and synthetic XML data::

1. Shakespeare collection: a set of 39 plays of Shakespeare in XML format<sup>7</sup>;
2. XBench Database Generator from Waterloo University<sup>8</sup>: XBench generates data-centric (e-commerce or transactional data) or text-centric (book collections or news articles) XML data in the form of either single or multiple documents, ranging from 10MB to 10GB in size. In the experiments, we tested both data-centric and document-centric single and multiple document mode, where the size was 10MB or 100MB.

For each tree  $T$  in a collection  $S$ , we simulate the authoring process as follows. We build the tree by appending nodes in the depth-first left-to-right traversal order. At each step of the traversal, the cursor points to position after the current node and is an object of suggestion. Only one suggestion ( $f_{opt}$ ) is considered, and if this suggestion matches the rest of the tree  $T$ , it is counted as a success. Two complementary measurements are used to evaluate the adviser performance on tree  $T$  or a test set. These measurements are the following:

1. Match ratio  $MR(T)$  for tree  $T$  is evaluated as  $m/|T|$ , where  $m$  is number of matches and  $|T|$  is the total number of nodes in the tree; it gives the ratio of successful matches on  $T$ . In the case of test set, the match ratio  $MR$  is an average of individual ratios  $MR(T)$  over all trees  $T$  in the set.
2. Gain ratio  $GR(T)$  for tree  $T$  is evaluated as  $\sum_i |f_i|/|T|$ , where  $|f_i|$  is the size of a successful suggestion  $f_i$  at step  $i$  of the traversal ( $|f_i|=0$  in the case of failed suggestion). In the case of a test set, the gain ratio  $GR$  is averaged over all trees in the set.

The match ratio measures the adviser effectiveness, while the gain ratio measures the adviser efficiency. The former measures merely the percentage of successful suggestions, so getting values between 0 and 1; the latter takes into account the suggestion size: the longer are correct suggestions, the higher is the gain ratio. For example, the gain value 4 indicates that the adviser reduces the cost of the document structure generation in 4 times.

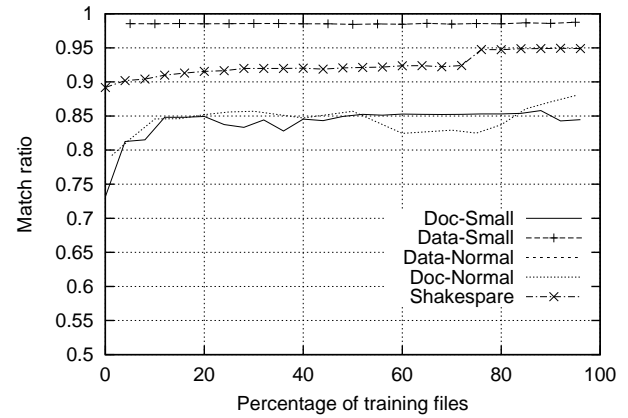


Figure 5: Match ratios in the off-line training.

## 7.1 Experiments

### 7.1.1 Off-line training

We have run the first series of experiments on four collections generated by XBench and the Shakespeare collection. With XBench generator, two small size (10MB) and two standard (100MB) collections were generated using data-centric and document-centric data. The collections are denoted as Doc-Small, Data-Small, Doc-Normal and Data-Normal. For each collection, we employ the *incremental off-line training* mode, when the adviser extracts suggestion patterns from  $p\%$  of documents in a collection and then applies them to  $(100-p)\%$  remaining documents, where  $p$  varies from 1% to 95%. We measure both match and gain ratios for all values of  $p$  and plot them in Figures 5 and 6. In all experiments, the context is minimal and the depth of suggestions is limited to  $d = 1$ .

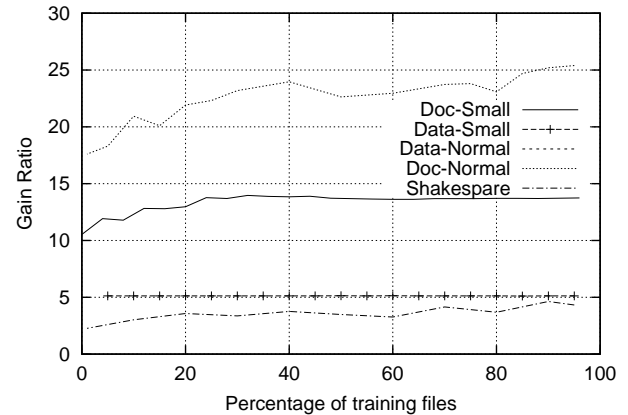


Figure 6: Gain ratios in the off-line training.

As Figures show, for data-centric XML, both match and gain ratios do not vary much (0.98 match ratio and 5.12 gain ratio), they merely coincide for both small and normal dataset size. For document-centric XML, the size does not influence the match ratio, while the gain ratio is considerably higher for the normal size than for the small case. For Shakespeare collection, the match ratio remains around 90%, and the gain ratio oscillates around 4.3 which is the lowest gain level achieved in the experiments.

<sup>7</sup>Version 2.0 of Shakespeare plays tagged with J.Bozak’s XML vocabulary is available from <http://xml.coverpages.org/>.

<sup>8</sup>Available from <http://db.uwaterloo.ca/dbms/projects/xbench/>.

### 7.1.2 On-line training

In the second series of experiments, the adviser is trained in the *incremental on-line* mode. With the help of the Xbench generator, we have generated one data-centric and one document-centric XML document, 10MB each. For Shakespeare collection, the experiment was run on each of 10 randomly selected documents. Each document in each experiment is traversed in the depth-first, left-to-right manner. At each step of the traversal, the adviser is trained with the already traversed fragment including the current node, and applies patterns to the cursor position pointing after the current node. Figures 7 and 8 plot match and gain ratios for the two Xbench documents and average values over selected documents in Shakespeare collection. For the data-centric Xbench document, it is sufficient a very small fragment to achieve the saturation point for both match and gain ratios. Considerably longer is the training for the Xbench document-centric document, but the slowest gain growth is with Shakespeare collection, though the match ratio saturates on average after 40%-50% of a document.

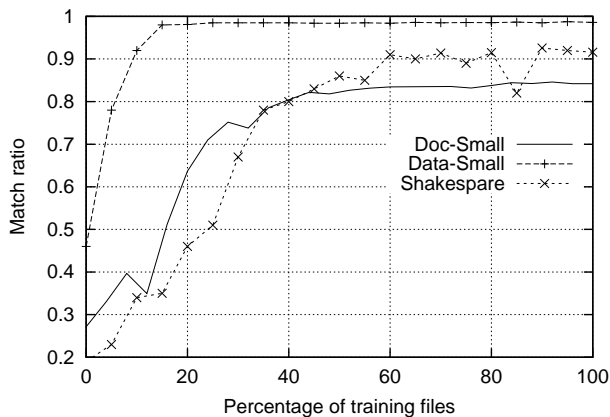


Figure 7: Match ratios in the on-line training.

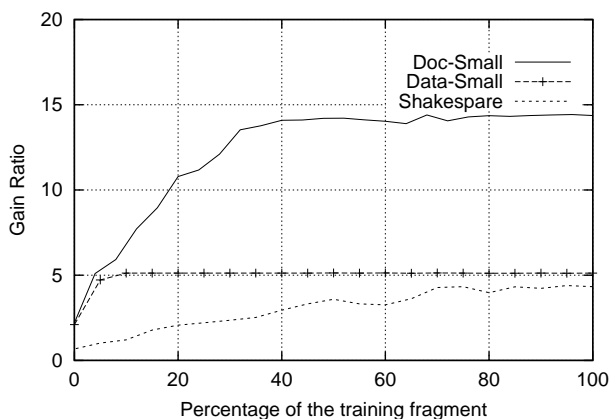


Figure 8: Gain ratios in the online training.

## 7.2 Discussion

The series of experiments conducted with real and synthetic data have shown very promising results for the data mining approach to the structural advising, as it allows to considerably reduce the cost of document structure generation (4.3 times for the real documents in Shakespeare collection). Below we discuss some limitations of the current approach, open questions and problems that will require further analysis and research, in order to further improve the adviser performance.

- The most obvious limitation of the current approach comes with the suggestion index developed in Section 5. The index essentially proposes data structures for suggestion candidates of depth 1 and is optimized to the append mode of document generation. A full management of tree patterns of depth 2 and more, as well as non-empty prefixes and suffixes, represents a very important issue for the performance of suggestion index and the adviser in general.
- The suggestion index has been tested in both off-line and on-line modes. Tests demonstrate that finding optimal patterns with the data structures in the index performs well in the off-line training. Once these structures (in the form of prefix automata) are built from sample data, they remain unchanged during the document edition process. Instead, the on-line training mode imposes additional requirements to the data structures, since the states, transitions and associated aggregate values in automata get updated after any authoring step. The current implementation of the index exposes certain limits and will require the design of the *incremental and dynamic* version of the data structures for the storage and retrieval of optimal suggestions.
- The adviser architecture and method address the finding of optimal structural patterns of *elements* for the XML document authoring. Clearly, the idea of finding analogies between a current document and sample data is not limited to elements only; it can be extended to other components of XML documents, including element *attributes*, *key dependencies*, etc.

We want to conclude this section by the remark that our study concerned primarily the data mining aspect of the structural advising for XML documents. The method we proposed here shows that mining available data can considerably increase the “intelligence” of an XML editor when assisting the authoring process. However, a number of important issues relevant to structural advising in an XML authoring system remained beyond the score of this paper. These issues like the graphical user interface or integration the data mining paradigm in the authoring environment put the user in the center of consideration. Recently, we have built a prototype that integrates the structured adviser in Adobe FrameMaker 6.0 in the form of plug-in. The next step will be developing different scenarios of providing structural suggestions to the user and running a set of evaluations through the case study and behavioral analysis.

## 8. CONCLUSION

In this paper, we have proposed a structural adviser for the XML document authoring problem. To our best knowledge, this is the first attempt to propose a method for mining available data and to rank tree patterns of different size accordingly to the efficiency metrics, expressed by the similarity and gain functions. The principle of an adviser and contextual suggestions is close to those implemented in various document editors, like MS Word, Emacs,

Amaya, for *spell-checking tasks*; the knowledge of the language and associated dictionaries are hard-coded in the editor. The difference is that suggestions in these editors cope with the content of document; while the suggestions patterns in our structural adviser try to capture the structure of a document; moreover the patterns can be identified off-line or from scratch in the on-line mode.

## 9. REFERENCES

- [1] "Corel XMetaL 4". <http://www.corel.com/>.
- [2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
- [3] "Xeena at IBM Alphaworks". <http://www.alphaworks.ibm.com/tech/xena>.
- [4] Donald D. Cowan, E. W. Mackie, G. M. Pianosi, and G. de V. Smit. Rita - an editor and user interface for manipulating structured documents. *Electronic Publishing*, 4(3):125–150, 1991.
- [5] Brian Dunkel and Nandit Soparkar. Data organization and access for efficient data mining. In *Proc. ICDE*, pages 522–529, 1999.
- [6] R. Furuta, V. Quint, and J. André. Interactively editing structured documents. *Electronic Publishing*, 1(1):19–44, 1988.
- [7] Ke Wang and Huiqing Liu. Discovering Structured Association of Semistructured Data. *IEEE Trans. Knowledge and Data Engineering*, 12(3):353–371, 2000.
- [8] Frank Neven. Automata Theory for XML Researchers. *SIGMOD Record*, 31(3):39–46, 2002.
- [9] "Microsoft XML Notepad". <http://www.notetab.com/>.
- [10] Nestorov S., J. D. Ulmann, L. Wiener, and S. S. Chawathe. Representative Objects: Concise Representations of Semistructured, Hierarchical Data. In *Proc. Intern. Conf. Data Engineering*, pages 79–90, 1997.
- [11] Mark Sifer, Yardena Peres, and Yoelle Maarek. "browsing and editing xml schema documents with an interactive editor". In *Proc. 3rd Workshop on Databases in Networked Information Systems (DNIS)*, 2003. University of Aizu, Japan.
- [12] "XML Spy". <http://www.xmlspy.com/>.
- [13] D. Wood. Standard Generalized Markup Language: Mathematical and philosophical issues. *Lecture Notes in Computer Science*, 1000:344–365, 1995.
- [14] Mohammed J. Zaki. Efficiently mining frequent trees in a forest. In *Proc. ACM SIGKDD*, pages 71–80, 2002.