

Graph Transformation to Infer Schemata from XML Documents

Luciano Baresi
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano, Italy
bares@elet.polimi.it

Elisa Quintarelli
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano, Italy
quintarelli@elet.polimi.it

ABSTRACT

Semi-structured data are characterized by the lack of a pre-defined schema. This heterogeneity simplifies the management of such data, but analysis and queries become more difficult and demand for schemata that describe these data. Super-imposed structures cannot be as general as predefined ones, but ease the retrieval of the information embedded in such data. The paper adopts XML as the language to render semi-structured data and proposes an approach – based on graph transformation techniques – to infer the schemata of XML documents.

Keywords

Semistructured Data, XML, DTD, Graph Transformation

1. INTRODUCTION

In these years, XML (eXtensible Markup Language, [13]) is used to render many different types of information (e.g., communication protocols and graphical formats). Among them, the database community concentrated on XML to represent and exchange *semi-structured data*, that is, data with no absolute and fixed schema and with a possibly irregular and incomplete structure [1].

These data store significant information, but its retrieval is hard. If we consider standard database technology, the formulation of queries is based on the availability of a schema, which is usually defined even before the data themselves. In contrast, semi-structured data do not usually show their internal structure, but its availability would definitively ease the formulation of queries.

In this paper, we assume the use of Document Type Descriptors (DTDs) to define the schema of XML documents and specify their structure. Unfortunately, many XML files do not come with their associated DTDs and the structure remains hidden. In some cases, this is not a problem, but in other situations it could really hamper the (semi)automatic

retrieval of the information embedded in the documents. For example, if we consider very large XML documents automatically generated from data stored in databases, repositories, or files [7], we could ignore the exact organization of the document. But when the document is the only source of information, the availability of a DTD plays an important role in the efficient management of the data and in the specification, optimization and processing of queries over the XML document [3]. This is the case of mediators that integrate heterogeneous documents: they use the DTDs to optimize the queries to data sources and retrieve the information that define the integrated documents [9].

Since DTDs are not always available, the capability of inferring them given simple XML documents is a key issue to manage and query these documents efficiently. The inference capability cannot produce the same DTD as if it were defined before editing the document, but it can return a “reasonable” approximation. The main difference is that if we think of a DTD and then produce XML documents, we start from the general definition of a family of documents and instantiate it to produce some specific documents. If we infer the structure from a document, or a set of documents, we can only be as generic as the documents from which we start.

This paper proposes a solution – based on graphs and graph transformation – to the inference problem. We transform XML documents into trees (graphs) where nodes render data and edges state the relationships among them. We adopt graph transformation rules to modify such trees to infer a graph-based notion of the schema with which the XML document comply. The graph-based representation is then serialized into the actual DTD. The paper presents the first results. It defines our notion of graph-based representation of XML documents and DTDs, describes the inference rules, and summarizes the first experiments.

The paper is organized as follows. Section 2 briefly introduces XML and the XML Infoset, and shows their corresponding graph representations. Section 3 presents and discusses our approach. Section 4 surveys related proposals and Section 5 concludes the paper.

2. XML AND THE XML INFOSET

XML datasets are often called *documents* because they can be serialized as plain text. However, unlike generic text documents, XML documents are not completely unstructured. An XML document is a sequence of nested elements, each delimited by a pair of start and end tags (e.g., `<tag>`

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.

SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA
Copyright 2005 ACM 1-58113-964-0/05/0003 ...\$5.00.

and `</tag>`). The sequence is itself enclosed within a *root element*. Fig. 1 shows a well-formed XML document used as example in this paper.

```
<CATALOG>
  <CD Code = "1" >
    <TITLE> Empire Burlesque </TITLE>
    <ARTIST> Bob Dylan </ARTIST>
    <COUNTRY> USA </COUNTRY>
    <COMPANY> Columbia </COMPANY>
    <PRICE> 10.90 </PRICE>
  </CD>
  <CD >
    <TITLE> Hide your heart </TITLE>
    <ARTIST> Bonnie Tyler </ARTIST>
    <COUNTRY> UK </COUNTRY>
    <COMPANY> CBS Records </COMPANY>
    <YEAR> 1988 </YEAR>
  </CD>
</CATALOG>
```

Figure 1: A well-formed XML document

The text-like representation of XML is used for many applications, but the XML standard data model, called *Infoset* [12], is not text-based. Rather, it represents both XML schemata and documents as *multi-sorted trees*, i.e., trees whose nodes (called *information items*) belong to a variety of types. In other words, the information set of an XML document consists of a number of information items. The information set for any well-formed XML document contains at least a document information item (the root of the tree) and several other items. An information item is an abstract description of some parts of an XML document: each information item has a set of associated named *properties*¹.

In this work we consider the *element-only* Infoset content model, which allows an XML non-terminal tag to include only other elements and/or attributes, while the text is confined to terminal elements. Furthermore, without loss of generality, we do not consider some features of the Infoset that are not relevant to this work, like namespaces.

The Document Type Descriptor (DTD) of an XML document (or a set of documents) is not mandatory and has the role of a schema specifying the structure of the document. In this work we consider how to infer a DTD from an XML document.

For example, Figure 2 presents an example of DTD for the XML document of Fig. 1. It is also the result of the inference process presented in this paper and explained in the next section.

XML Tree. Following the Infoset conventions, we represent an XML document by means of a labeled tree, called **XML tree**, $\langle N, E, r, \ell \rangle$ where N is the set of nodes, $r \in N$ is the root of the tree (i.e. the root of the XML document), E is the set of edges, and ℓ is a labeling function: $\ell : N \rightarrow \mathcal{L} \cup \{\perp\} \times \{\text{root, element, text, attribute}\} \times \text{PCDATA} \times \mathbb{N}^4$, where \mathcal{L} is a set of labels and \perp means ‘undefined’. ℓ can be seen as the composition of the seven single-valued functions ℓ_N , ℓ_T , ℓ_V , ℓ_{P_1} , ℓ_{P_2} , ℓ_{C_1} , and ℓ_{C_2} . Moreover, the following property on nodes holds: each node n_i has a

¹In the Infoset specification, property names are shown in square brackets, [thus]. We do not follow this convention in this paper.

```
<!DOCTYPE CATALOG [
  <!ELEMENT CATALOG(CD)+>
  <!ELEMENT CD(TITLE,ARTIST,COUNTRY,COMPANY,
    PRICE?,YEAR?)>
  <!ATTLIST CD Code #IMPLIED>
  <!ELEMENT TITLE(#PCDATA)>
  <!ELEMENT ARTIST(#PCDATA)>
  <!ELEMENT COUNTRY(address)>
  <!ELEMENT COMPANY(#PCDATA)>
  <!ELEMENT PRICE(#PCDATA)>
  <!ELEMENT YEAR(#PCDATA)>
]>
```

Figure 2: The DTD derived for the XML document of Fig. 1

tuple of labels $\ell(n_i) = \langle Ntag_i, Ntype_i, Ncontent_i, NMinPos_i, NMaxPos_i, NMinCard_i, NMaxCard_i \rangle$, where $Ntag_i$ is the name of the element (or attribute)², $Ntype_i$ indicates whether the node type is root, element, text, or attribute, and $Ncontent_i$ assumes either a PCDATA or \perp (undefined) as value. $NMinPos_i$ and $NMaxPos_i$ are used to represent the position (start and end positions) of a node among the set of children of a chosen node, whereas $NMinCard_i$, and $NMaxCard_i$, are introduced to store the minimum and maximum number of occurrences of a given node. Cardinality labels on nodes are used during the graph transformation process to store the cardinality of each node. These labels are almost useless for the tree-based representation of an XML document, but they are used in the tree-based notion of the schema.

Moreover, the following constraints hold on an XML tree $\langle N, E, r, \ell \rangle$: 1) $\ell_T(r) = \text{root}$. In an XML document the root node has type label equal to root. 2) $\forall n \in N$ it holds that $\ell_T(n) = \text{element} \rightarrow \ell_V(n) = \perp$ and $\ell_T(n) = \text{text} \rightarrow \ell_N(n) = \perp$. The content label of element nodes is undefined and the tag label for text nodes is not explicitly specified. 3) $\forall n \in N$ it holds that $\ell_{C_1}(n) = 1$ and $\ell_{C_2}(n) = 1$. In an XML tree representing a document, the number of occurrences of each node is set to 1.

The XML tree of the document of Fig. 1 is presented in Fig. 3. Notice that we have chosen to represent elements and attributes in a uniform way (i.e. with the same tuple of labels), thus, also attribute nodes have position and cardinality labels.

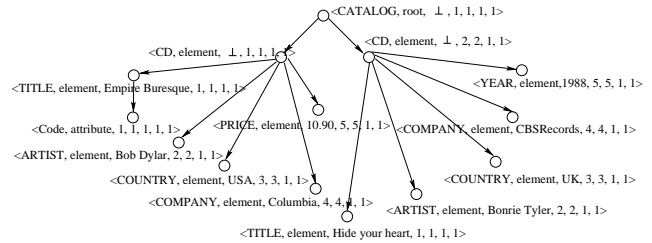


Figure 3: The XML graph for the document of Fig.1

²To uniquely identify a node one should introduce the notion of Node Identifier (a sort of Object Identifier) or use the path, composed by tags, from the root to the node as node name.

3. APPROACH

This section describes our approach for inferring a “reasonable” DTD given an XML document. The presentation starts by introducing graph transformation and AGG (the tool used for our experiments), then describes XML Schema Trees, and concludes with the actual inference rules.

Graph Transformation. Graph transformation systems are our means to specify inferences rules. Specifically, a *typed graph transformation system* $\mathcal{G} = \langle TG, C, R \rangle$ consists of a type graph TG , a set of structural constraints C over TG , and a set R of rules $p : L \Rightarrow R$ over TG . A graph transformation rule $r : L \Rightarrow R$ consists of a pair of TG -typed instance graphs L, R such that the intersection $L \cap R$ is well-defined (this means that, e.g., edges which appear in both L and R are connected to the same nodes in both graphs, or that vertices with the same name have the same types, etc.). The left-hand side L represents the pre-conditions of the rule while the right-hand side R describes the post-conditions.

The application of a graph transformation rule comprises three steps: 1) We find an occurrence o_L of the left-hand side L in the current graph G (the XML document, in our case). 2) We remove all the vertices and edges from G which are matched by $L \setminus R$. The remaining structure $D := G \setminus o_L(L \setminus R)$ must be a legal graph: no edges are left dangling because of the deletion of their source or target vertices. In this case, the *dangling condition* is violated and the application of the rule is prohibited. 3) We glue D with a copy of $R \setminus L$ to obtain the derived graph H . We assume that all newly created objects, links, and attributes get fresh identities, so that $G \cap H$ is well-defined and equal to the intermediate graph D .

In this work, we use *AGG* (Attributed Graph Grammars, [2]) as supporting tool because of its Java-like syntax (for specifying the type graph and attribute values) and the availability of a stand-alone execution engine. AGG allows users to specify complex structures as graphs and exploits the Java type system to associate attributes with values. It also supports *layered rules*. Layers define the order to apply rules. The interpretation starts with level-zero rules and then moves to higher ones. Besides pre- and post-conditions (left- and right-hand side graphs), AGG also supports negative application conditions, that is, sub-graphs in the left-hand side that must not exist to enable the rule. Additionally, rules can embed conditions on attribute values in the form of boolean Java expressions.

XML Schema Tree. We introduce the definition of **XML Schema Tree** to avoid confusion with the notion of XML schema of a document, which is an XML document itself representing the structure of the document. An XML Schema Tree is an XML tree $\langle N, E, r \rangle$ such that each node $n_i \in N$ has the content label $Ncontent_i$ undefined. Moreover, each tag name (actually, the path name) has at most one occurrence in the tree.

REMARK 1. *If the XML tree that represents a document has more than one occurrence of a given tag name in different parts of its structure (i.e., in different paths of the tree), then the tag label of each node is substituted by the path name.*

Given an XML document D , we indicate with $\mathcal{S}(D)$ its XML Schema Tree. For example, if D were the document

of Fig. 1 its $\mathcal{S}(D)$ is reported in Fig. 4: the node content labels are undefined (i.e. their value is \perp).

Since in this work we infer the XML Schema Tree of a document by the document itself, the tree-based structure does not always correspond to the original DTD of the document (if available). In particular, it may differ from the original DTD in the specification of the cardinalities, and in the order of elements, and it could not mention some elements (i.e., the elements that are present in the DTD but not in the document). Notice that these are not limitations of the approach, but there are inherently part of the inference process itself. If the original DTD contained cardinalities wider than their instantiations in the document or if it allowed elements that are not used in the document under analysis, we cannot infer them: we do not have enough information to do this. Moreover, the order among different elements and sequences of elements are established only by analyzing the document (i.e., a particular instance of the original DTD).

Transformation rules. In this paper we want to compute $\mathcal{S}(D)$, given an XML document D , by using graph transformation rules and derive the DTD for D in such a way that D is valid with respect to the inferred DTD.

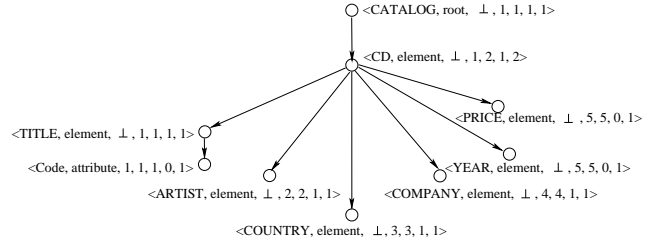


Figure 4: The XML Schema Tree of the document of Fig.1

Given the XML Schema Tree $\mathcal{S}(D) = \langle N, E, r \rangle$ of a document D , the corresponding DTD is obtained by a depth-first visit of the tree and by interpreting the cardinality labels as follows. Given a node $n_i = \langle tag_i, type_i, content_i, MinPos_i, MaxPos_i, MinCard_i, NMaxCard_i \rangle \in N$ then it holds that:

- if $type_i = \text{attribute}$, then: (a) if $MinCard_i = 0$ then the attribute is **IMPLIED**, (b) if $MinCard_i = 1$ then the attribute is **REQUIRED** (we do not consider attributes with **FIXED** (constant) value).
- if $type_i = \text{element}$, then: (a) if $MinCard_i = 0 \wedge MaxCard_i > 1$ then the element has zero or more occurrences (represented with the symbol $*$). (b) if $MinCard_i = 0 \wedge MaxCard_i = 1$ then the element is optional (represented with the symbol $?$). (c) if $MinCard_i = 1 \wedge MaxCard_i > 1$ then the element has one or more occurrences (represented with the symbol $+$). (d) if $MinCard_i = 1 \wedge MaxCard_i = 1$ then the element has exactly one occurrence.

The position labels are used to find out the order among the set of children of a given node by analyzing the $MinPos$ labels in ascending order.

These considerations guided us to devise the rules of Fig. 5 to automatically derive the XML Schema Tree of (any) XML

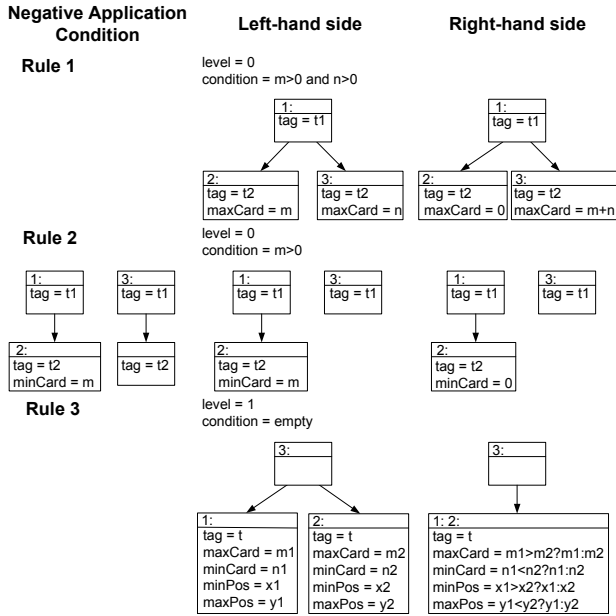


Figure 5: The transformation rules of our *inference engine*

document. These rules are the core of our *inference engine*. Since we conducted our experiments by using AGG [2], rules are presented using its syntax. Fig. 5 uses conditions to define the constrains that must hold to apply the rule, Java-like expressions to specify how the rule modifies attributes, and shared identifiers to mean that elements are preserved after applying the rule. The first two rules (level 0) work on the cardinalities associated with each node. The first rule counts the occurrences of a node in a sequence by summing the cardinalities of the two nodes. It also changes one of the cardinalities to 0 to avoid applying the rule infinitely (since the rule is applied only when the two cardinalities are greater then zero).

The second rule identifies optional elements. It sets the min cardinalities to zero to mean that the element is optional. Notice that we do not need to relate the two nodes to the same parent since they are uniquely identified by their tags (or path from the root). This rule can only be applied if the negative condition is satisfied, i.e., the second parent node has no child nodes with the same tag as the other child node and the min cardinality of the child node that must exist is greater than zero. This last constraint is to avoid infinite loops and ensure the termination of the derivation process.

After applying the first two rules and when we have no further possibilities for these rules, we can move to the third rule. It simply unifies the instance nodes to make them become unique and thus become nodes that render types. It also sets cardinalities and positions to identify optional and mandatory types, along with their multiplicities, and the sequences among tags. The rule computes *min* and *max* values by using the statement $\langle expr \rangle ? \langle trueval \rangle : \langle falseval \rangle$, where $\langle trueval \rangle$ and $\langle falseval \rangle$ are the attribute values if $\langle expr \rangle$ evaluates to true or false, respectively.

We started working on the validation of the approach by

building a first prototype environment implemented in Java.

Transformation rules were applied by using two different approaches: 1) they were encoded directly in Java to define the *inference engine*; 2) AGG was used as customizable *inference engine*. The main difference between the two solutions is performance; AGG is useful during the validation process when rules are modified frequently.

These prototype tools allowed us to carry out some experiments to infer the DTD for documents of up to 4 MBytes.

For example, if we considered our running example reported in Fig. 1, the tool first produces the tree-based representation of the XML document. The XML Schema Tree is obtained by iteratively applying the graph rewriting rules. The next figures present an excerpt of the transformation process. Notice that black dots represent the nodes that are modified by applying the rule.

For each node, the first transformation rule counts the number of occurrences of its sub-elements. In our example it only modifies the label *MaxCard* of CD elements (see Fig. 6).

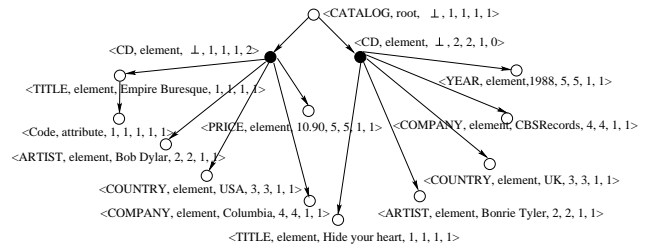


Figure 6: An application of the First Transformation Rule

The second transformation rule sets the label *MinCard* of optional elements or attributes. In our example, it sets the label *MinCard* to 0 for *YEAR* and *PRICE* elements, and for the *Code* attribute (see Fig. 7).

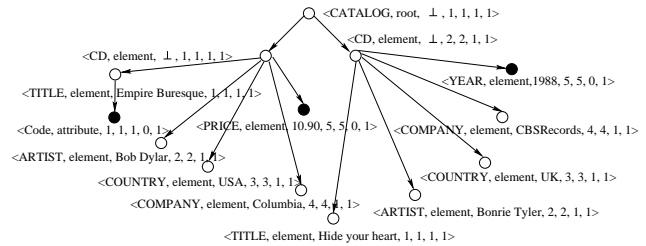


Figure 7: An application of the Second Transformation Rule

To conclude, the third rule is iteratively applied to unify multiple occurrences of a given node. For example, if we suppose it starts from the two CD elements, it produces the tree of Fig. 8. After this transformation, the first rule can be applied again to the set of children of the new unique CD element and the third rule can then unify *TITLE* elements and all the other children of the CD element (the resulting schema is in Fig. 4).

In the end the tool returns the tree-based representation of the Schema and produces the DTD.

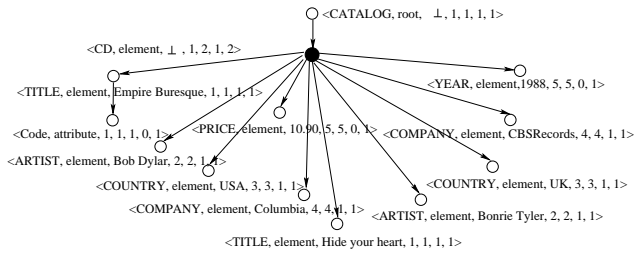


Figure 8: A first application of the Third Transformation Rule

4. RELATED WORK

The inference of DTD specifications from XML documents is a complex problem because the DTD syntax has the full expressive power of regular expressions. The analysis of the huge amount of work on the general problem is out of the scope of this paper. Here, we limit our attention to semi-structured data.

The problem of extracting a schema from semistructured data has been developed in [4, 6, 8], but the kind of schema-structure extracted in these works is quite different from DTDs. Moreover, in [10] the authors describe RoadRunner, a system for inferring wrappers for structurally similar pages. The RoadRunner approach is based on the representation of wrappers as union-free regular expressions. It mainly consists of parsing a source HTML document by using the wrapper and refining it whenever a mismatch occurs. The limit of this work is that a wrapper can be inferred only for pages that comply with a union-free regular grammar.

To the best of our knowledge the problem of inferring DTDs from a collection of XML documents has been addressed in [9, 5]. In [9], the authors study the inference of DTDs from views of XML data. In particular, they extend the DTD declaration syntax in order to infer tight DTDs that characterize a selection view on XML sources. Views are specified by a query language that produces a list of documents selected from one or more inputs sources; thus, in this work the authors does not study how to determine the DTD starting from XML data. The problem of inferring the DTD directly from XML documents is addressed in [5]. The authors describe the XTRACT inference algorithm, which comprises three steps: 1) finding patterns in the input documents and replacing them with regular expressions to generate candidate DTDs, 2) factoring the candidate DTDs by using logic techniques to compact them, 3) applying the Minimum Description Length (MDL) principle to find the most compact and precise DTD among the set of candidates.

We believe that graph transformation techniques allow one to take advantages from the natural tree-based representation of XML documents to abstract both concise (i.e. small in size) and precise schema graph-based structures, which can be used to derive DTDs of XML documents.

5. CONCLUSIONS AND FUTURE WORK

The paper has presented an approach based on graph transformation for inferring the schemata of XML documents. It has also sketched the first version of the prototype tool. We are pretty confident that the inference rules defined so far are enough to address a wide class of XML documents,

but we need more experiments to fully qualify the class of considered documents. Even if we compare our approach with other proposals in the field, we see interesting and novel aspects not touched by the other approaches. First of all, we propose the same data structure, a labeled tree, to represent both XML documents and schemata. Thus, the XML Schema Tree of a document can also be seen as an XML document specifying the structure of one (or more) data sets.

These promising results are paving the ground to our future work, whose aim is mainly the consolidation of the approach and the extension of the experimental validation. Term graph rewriting need further attention, but we can also easily foresee the improvement of our prototype tool and the adoption of XML Schema [11] as alternative to DTDs.

6. REFERENCES

- [1] S. Abiteboul. Querying Semi-Structured Data. In *Proceedings of the International Conference on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, pages 262–275, 1997.
- [2] The Attributed Graph Grammar System. <http://tfs.cs.tu-berlin.de/agg>.
- [3] D. Braga and A. Campi. A graphical environment to query xml data with xquery. In *Proceedings of 4th International Conference on Web Information Systems Engineering*, pages 31–40. IEEE Computer Society, 2003.
- [4] M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proceedings of the Fourteenth International Conference on Data Engineering*, pages 14–23. IEEE Computer Society, 1998.
- [5] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. Xtract: Learning document type descriptors from xml document collections. *Data Mining and Knowledge Discovery*, 7(1):23–56, 2003.
- [6] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, pages 436–445. Morgan Kaufmann, 1997.
- [7] L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a first study. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 500–510. ACM Press, 2003.
- [8] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 295–306. ACM Press, 1998.
- [9] Y. Papakonstantinou and V. Vianu. Dtd inference for views of xml data. In *Proceedings of 19th Symposium on Principles of Database System*, pages 35–46. ACM Press, 2000.
- [10] G. Mecca V. Crescenzi and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the Twenty-Seventh International Conference on Very Large Data Bases*. IEEE Computer Society, 2001.
- [11] World Wide Web Consortium. XML Schema, 2001. <http://www.w3c.org/TR/xmlschema-1/>.
- [12] World Wide Web Consortium. XML Information Set, 2001. <http://www.w3c.org/xml-infoset/>.
- [13] World Wide Web Consortium. Extensible Markup Language (XML) 1.0, 1998. <http://www.w3c.org/TR/REC-xml/>.