

Study of the Automatic Construction of XML Documents Models from a Relational Data Model

F. Laforest

LIRIS, INSA - bat B.Pascal - 69621 Villeurbanne cedex - France
frederique.laforest@insa-lyon.fr, mboumedi@lisi.insa-lyon.fr

M. Boumédiene

Abstract

End-users information capture remains a sensitive challenge, especially when information is under the form of documents. The difficulty concerns information indexing so that information can be precisely queried. In the DRUID project, the end-user captures XML paragraph-centric documents (i.e. documents with tags delimiting narrative text paragraphs), and a transformation tool generates XML data-centric documents that can be stored in a database. The database can then be queried to retrieve documents. Data-centric and paragraph-centric documents conform to DTDs. In this article, we present algorithms to automatically generate DTDs from the database schema.

1. Introduction

A document-based user interface is not dedicated to all kinds of end-users. But a lot of domains would prefer using them rather than forms. One can cite doctors and physicians who write notes about their patient, lawyers who prepare a client file, journalists who could write articles and retrieve others thanks to automatic indexing.

Using input forms allows to store data in a database. This approach is very effective for queries, but data input is heavy. Another approach concerns free text input in documents, stored in a documentary fund. Information capture is highly flexible but querying is not as powerful as with database. Other approaches exist between these two extremes, such as structured documents. A structured document contains a set of typed and organized information. In other words, structured documents contain information and semantics guides (also known as logical structure). These guides are most of the time represented by tags circling information chunks. XML documents are good examples of structured documents[1]. A Document Type Definition (DTD) provides tags, composition rules... This structure is not as rigorous as forms structure: no

format is imposed for data. Queries on structured documents can be made using dedicated query languages [2][3]. One can find two types of structured documents:

- In data-centric documents (DCD), each information is tightly tagged so that there is a one-to-one relationship between data in the database and tagged information in documents. This kind of document does not satisfy all end users, as writing such documents is time consuming and much constraining.
- In paragraph-centric documents (PCD), most tags delimitate paragraphs rather than data. A paragraph contains free text which may include many data. For example, a prescription paragraph contains one sentence in which one can read a medication name, a dose, a frequency and a duration. This information is drown in the prescription sentence. PCD have a double advantage: (1) to be easily captured and read by the end-user and (2) to be semantically more precise than free text, thus easier to analyze automatically.

The main idea of our global project DRUID (document and rule-based user interface for database) is to use paragraph-centric documents as a user interface paradigm, and calculated data-centric documents as an intermediate format for relational data storage. The model of the database is domain-related. It does not contain generic tables like “node” or “attribute”, but a classical domain-related model, in which we have only added captured documents URLs. Captured documents are stored on the file system of the server. Using a relational database allows for precise queries: they are made on the database, answers include data in the database and links to the corresponding captured documents. We have developed a tool that makes the necessary transformations between captured paragraph-centric documents and data-centric documents. These transformations are based on domain-related, XSLT-like transformation rules. For more information on the transformation tool, one can read [4][5].

PCDs and DCDs are all based on DTDs deduced from the database model. For one database model, many documents types are required. For example, in a patient

record application, we need documents for physicians, others for nurses, and so on. A physician may once need a document type to write surgery reports, and another type to write prescriptions... A DTD has to be written for each context of use. This fact can be compared to the menus a classical form-based capture software proposes.

In this article, we present our algorithms that generate automatically DTDs from a database model. The DCD DTDs have close relationships to the database, while the PCD DTDs are a simplification of the DCD DTD. We have defined new algorithms that have the same basis as the work from Pr. Bourret [6], but include other considerations and constraints, that we explain hereafter.

The structure of this article is as follows. After a tour of related works, the first point concerns data-centric documents DTDs: the DCD DTD is directly deduced from the database model. The second section concerns PCD DTDs, built from the previous DTD. We conclude our work and provide future directions of work.

2. Related works

With the arrival of XML, querying structured documents has become a major research topic. There are two main research directions. The first uses languages dedicated to XML documents, like XML-QL [7] and Lorel [8]. They are mainly dedicated to data-centric documents. The second concerns the management of documents in databases, as presented in this section. There are three main techniques to store XML documents in a database:

- The first one stores documents as CLOB attributes. Some softwares like [9] propose this method. Searching for documents can be done in two ways : queries on context (other columns of the table), or full text search.
- The second one is based on the tree representation of documents [10][11]. It is based on generic tables like “edge”, “node”, “attribute” and can store any structured document. But querying is heavy: a lot of joins must be done on tables even for simple queries. Moreover, in paragraph-centric documents, joins have to be redefined: what does it mean to joins with a long character set column? Answering documents is also a long task, as documents have to be compound from tables. [9] proposes to store documents as objects where each object is a XML tree node; one must be informed of the document structure to query it.
- The third one is based on documents DTDs. The database model is directly related to the DTD structure, and only documents conforming to this structure can be inserted in the database. There are a lot of variations around this in the literature. Some build the database model from the DTD, others build the DTD from the database model. Some studies also try to allow for the

integration of documents of different DTDs. Let us see these works in more details.

From a DTD to a database model. [6] proposes an algorithm to generate a database model from a DTD. This algorithm must be seen as a first step, it does not take into account all it should. For example, multivalued attributes and the ‘|’ (or) operator are not treated. In [12] they transform a DTD into a relational schema. During the transformation, they transcript explicit semantic constraints into the relational schema. In [13] an algorithm allows to map a DTD to a database. They first assign a unique id for every object, and then normalize the 1-n relationships using two tables, one for the container and the other one for the contained object. In [14] documents are stored in a relational database where each level of the tree becomes a table in a relational schema. The first level in the document becomes the base table in the resulting schema. This table is seen as a fact table in the datawarehouse star schema. This table contains the basic identification data for the document object. All second levels are satellite tables. [9] provides a technique based on the creation of a particular data types for each tag. No full text search is allowed. However all XML structures cannot be mapped easily into database structures. Their mapping brings to document fragments. To remedy this problem, work [15] consists in using a hybrid database. XML documents are stored in an object-relational database, with a specific complex attribute of type XML. For example, in libraries, structured data (called regular information) are the title, the author, the editor, etc. while the document (irregular information) is in the same database, under the form of a dedicated XML object type.

From a database model to a DTD. In another context other researchers were interested in the DTD generation from databases, but we found few works concerning this domain. [16] presents an algorithm to get a DTD from a conceptual data model. The deduced DTD concerns the whole database. In [17] authors present a SGML DTD generation method using UML diagrams. Another proposition has been exposed in [6]. This algorithm is certainly effective but it presents many limits, as cardinality is not well managed (cardinality + or 1).

3. Data-centric documents

There are narrow ties between DCDs and the database. All information that must be stored in the database are first in DCDs. Each information unit for the database is demarcated so that there is a one-to-one relationship between tagged data in DCD and data in the database. Documents models and database model have thus a tight relationship.

3.1. Functional constraints on DCDs

Width of a DTD. Different types of users may write different types of paragraph-centric documents. We cannot provide only one type of document that would permit to fill all data foreseen in the base: the user would be lost in the number of available tags. End users way of working is more contextual, and most of the time correspond to only a sector of the database. We therefore propose to write one document type for each sector of the database. Each type of document concerns only a part of the database model. The designer chooses a table of the model as the main topic of the DTD. The name of this table is the root element of the DTD. Traversal of its columns and related tables allow for the definition of other elements in the DTD. Choosing a root table provides a limit “in width” of the database model.

Depth of the DTD. Reference links in the database model allow the navigation from table to table. In each document, the traversal of these links should not be exhaustive. The user would be lost in information capture. For example, in a teaching department, the “level of study” table contains a reference to the list of “courses” given at this level, each course contains a reference to the “teacher” organizing this course. Inserting a new level of study in the database should provide information for the “level of study” table, provide values for the “courses” list including the organizing teacher. But this document will not ask for all information for the “teacher” table. The data capture logic drives to a limit in reference links traversal. This is what we call a depth limit for the DTD. In our prototype, we have chosen to automatically limit reference links development to depth 1, but the DTD designer can change this value for each link.

XML element or attribute. Columns in the database can be represented as elements or attributes in documents. In HTML, element values are always shown to the end-user, while attributes are hidden. They are used by the browser for specific presentation rules or actions. For example, in `foo`, the foo word is presented to the end-user: it is the content of the element. But the `http://www.foo.fr` value of the HREF attribute is not shown to the end-user but used by the browser. The same kind of rule is used in form-based user interfaces where labels are presented to the end-user even if each label is tightly associated to a hidden id value. We thus have chosen the following rules. Columns of the database that represent identifiers are inserted in DTDs as attributes. So they are hidden. The effective label shown is another column of the table, chosen by the designer.

3.2. Translation of a relational schema into a DCD DTD

- The root element of the DTD is the main subject of the document. It is a table of the database model, as chosen by the designer.
- Standard columns of the main subject table of the database are translated as elements in the DTD. They are sub-nodes of the root node.
- The primary key columns of the root table are translated as attributes of the root element.
- Foreign key columns that are included in the primary key have a particular treatment: a new sub-element is created, it has the name of the referenced table. If the depth is limited at this level, this element has PCDATA type. Otherwise, the referenced table has to be treated with our algorithm to define its composition.
- The main subject table may be referenced in other tables as a foreign key. In the document world, this should be translated as a set of sub-elements. For example, in a doctor table, there is a serviceId column that references the service in which the doctor works. Writing a document about a service may require to add the list of doctors who work in it. Our algorithm thus searches for tables in which the table primary key is referenced, and adds a sub-element for each of such tables. Cardinality of such elements is *.
- Foreign key columns that do not belong to the primary key are translated as new sub-elements, that have to be treated by this algorithm to define their composition. Cardinality of these elements is 1 or ? depending on the not null constraint.

<p>Step 1: Ask the designer for the root table Step 2: Build a root element carrying the table name. Step 3: All columns of the table that are not primary keys nor foreign keys, are sub-elements of the one created in step 2. Step 4: All new elements get type PCDATA Step 5: Add each primary key column as an attribute of the element created at step 2 Step 6: For each foreign key that is in the primary key and not in parent element, a) ask the designer for the level of development and which column of the linked table should be used as label b) prepare a PCDATA element containing the chosen column Step 7 : Put cardinality « ? » to new elements that can be null Step 8 : For the whole primary key, a) search for tables in which this primary key has been exported as a foreign key b) add sub-elements carrying the names of the tables found to the element created in stage 2 c) each of these elements gets cardinality " * " Step 9 : For each foreign key that is not primary key, a) get the table that contains this key as primary key b) add a sub-element carrying the name of this table to the element created on step 2 c) provide cardinality "1" or "?" if compulsory or not Step 10 : Redo steps 3 to 9 for each generated element.</p>

Here is the logical model of our database on which we are going to step our algorithm (underlined columns are primary keys, italics columns are foreign keys to the table indicated as indice):

```
SERVICE(serviceId, specialty)
DOCTOR(doctorId, servicIdservice, doctorName, firstName,
phone)
DISEASE(Id, label)
MEDICATION(medicationCode, medicationName)
PATIENT(patientId, SSnb, sex, patientName, firstName,
address, phone, birthdate)
PASTHISTORY(patientIdpatient, diseaseCodedisease,
familyConnection, date)
ENCOUNTER(place, problemdisease, doctorIddoctor,
patientIdpatient, date)
PRESCRIPTION(doctorIdencounter, patientIdencounter, dateencounter,
medicationIdmedication, qty, qtyUnit, freq, freqUnit, duration,
durationUnit)
```

Figure 1 present an example of a DCD DTD generated from this database model. The theme of the document is table « Encounter ».

```
<?xml version = ' 1.0 ' encoding = ' ISO-8859-1 '? >
<!ELEMENT encounter (place?, disease?, doctorName,
patientName, prescription*) >
<!ATTLIST encounter patientId #REQUIRED >
<!ATTLIST encounter doctorId #REQUIRED >
<!ATTLIST encounter date #REQUIRED >
<!ELEMENT place (#PCDATA) >
<!ELEMENT disease (#PCDATA) >
<!ATTLIST disease id #REQUIRED >
<!ELEMENT doctorName (#PCDATA) >
<!ELEMENT patientName (#PCDATA) >
<!ELEMENT prescription (qty?, qtyUnit?,
medicationName, freq?, freqUnit?, duration?, durationUnit?) >
<!ATTLIST prescription medicationCode #REQUIRED >
<!ELEMENT qty (#PCDATA) >
<!ELEMENT qtyUnit (#PCDATA) >
<!ELEMENT medicationName (#PCDATA) >
<!ELEMENT freq (#PCDATA) >
<!ELEMENT freqUnit (#PCDATA) >
<!ELEMENT duration (#PCDATA) >
<!ELEMENT durationUnit (#PCDATA) >
```

Figure 1: A DCD DTD

4. Paragraph-centric documents

Documents are data-centric to fill the database. But the end-user would be fed up with tagging if he had to put all tags. We propose to capture information under the form of paragraphs containing free text and to tag each paragraph. Such documents are called paragraph-centric documents. Each captured paragraph contains information on an homogeneous topic and is independent from other paragraphs. Removing a paragraph from a document does not disturb other paragraphs understanding. For example, during an

encounter between a patient and a doctor, the practitioner writes information concerning the patient reason for the visit in a “problem” paragraph. Drug prescription is the object of another paragraph. If the doctor removes the problem paragraph, prescriptions are still understandable. Paragraph-centric documents have a double advantage [5]. The end-user reads or edits them much easier than forms or data-centric documents. Information extraction is more efficient than from free text. A paragraph-centric document conforms to a PCD DTD. This DTD has the following characteristics.

4.1. Functional constraints on PCD DTD

A reduction of the corresponding DCD DTD. The PCD DTD is deduced from the DCD DTD. It is a simplification of the DCD DTD, from which some tags are removed. The PCD DTD has depth 1: the root element contains other elements which are all PCDATA. In other words, all sub-element of the root element are leaves of the document model. Building the PCD DTD requires two steps. First remove all elements of level 2 or more and replace them by PCDATA in their parent element. Second add a “free comment” tag for all paragraphs the end-user does not want to tag explicitly (or has not tagged yet).

Unordered documents. DRUID allows to capture paragraphs without any order constraint. The end-user can write paragraphs in the order of his thinking, without any regard on the DTD ordering of paragraphs. It provides another freedom degree to the user. “No matter on what the system wants, I write what I think and I tag with the paragraph tags it provides.” This bends XML rules: in XML, an element contains a ordered list of elements. The expression of unordered sets is possible in SGML with the ‘&’ separator. But it infers complex parsing algorithms to validate documents. That is the reason why this option is not proposed in XML. There is no straight way to express an unordered set of elements while preserving cardinality constraints in XML. Figure 2 shows an XML element definition including ordered sub-elements. It also shows the same element composition without ordering constraints on sub-elements. This example shows how heavy it is.

```
Element E contains an ordered list of zero or one A element,
zero or more B elements then zero or more C elements:
<!ELEMENT E(A+, B *, C *)>
The same E element but without any ordering of A, B and C
elements
{<!ELEMENT E ((B|C)*, A, (B|C)*)+ >}
```

Figure 2: Disordering a DTD

As a paragraph-centric document written by the end-user can be unordered. DRUID reorders the document paragraphs according to the DTD ordering specification before providing it to the parser for validation. This

technique permits to keep using standard parsers without requiring too heavy operations.

A document under edition. PCD documents are written by users. The user tags paragraphs when he wants in the writing process. He can write all free text paragraphs and then tag each one, or tag each paragraph just after writing it. The user may also write paragraphs that do not have any relationship with the database. To manage all that, the PCD DTD contains a specific tag that does not come from the DCD DTD. It is the “parag” tag.

Figure 3 provides the PCD DTD calculated from the DCD DTD given on Figure 1.

```
<?xml version = ' 1.0 ' encoding = ' ISO-8859-1 '? >
<!ELEMENT encounter (place?, disease?, doctorName,
patientName, prescription*, parag*) >
<!ATTLIST encounter patientId #REQUIRED >
<!ATTLIST encounter doctorId #REQUIRED >
<!ATTLIST encounter date #REQUIRED >
<!ELEMENT place (#PCDATA) >
<!ELEMENT disease (#PCDATA) >
<!ELEMENT doctorName (#PCDATA) >
<!ELEMENT patientName (#PCDATA) >
<!ELEMENT prescription (#PCDATA) >
<!ELEMENT parag (#PCDATA) >
```

Figure 3: A PCD DTD

5. Conclusion and perspectives

We proposed a solution that permits to automate the generation of DTDs from a database model for information capture and database feeding. Two types of DTDs are generated : data-centered DTDs allow for a direct correspondence between data in the document and columns in the database, while paragraph-centric documents provide freedom to the end-user as capture concerns paragraphs of free text. We achieved a prototype that permits to generate a general DCD DTD from an existing relational database. We used the Java language and the JDBC package. We also used a DTD package [18]. This package provides a parser and a set of classes that model the DTDs as Java objects. Our tests have been made on a medical database stored with the Cloudscape DBMS. In the near future we intend to improve our work by studying possibilities to minimize designer interventions in the DTD definition process. For example, the definition of foreign key traversal level should be studied.

6. Bibliography

[1] N. Bradley, The XML Companion, Addison-Wesley, 1999
[2] Lemaitre J., Muriasco E., Rolbert M. From Annotated Corpora to Databases : the SgmlQL Language, in Linguistic Databases, CSLI Lecture Notes n° 77, J. Nerbonne (Ed.), 1997, pp. 37-58.

[3] P. Buneman, M. Fernandez, D. Suciu. UnQL: A query language and algebra for semistructured data based on structural recursion, VLDB Journal, p. 76-110, 2000, also available at : .
[4] Y. Badr, F. Laforest, A. Flory DRUID : coupling user written documents and databases. ICEIS 2003, Angers, France, april 2003
[5] Laforest F., Flory A. Using Weakly-Structured Documents to Fill in a Classical Database. Journal of Database Management, Apr-Jun 2001:3-13
[6] Bourret R. Mapping DTDs to Databases, Xml.com journal, May 09, 2001.
[7] Deutsch A., Fernandez M., Florescu D., Levy A., Suciu D. XML-QL: A Query Language for XML. Submission to the World Wide Web Consortium 19-August-1998.
[8] Jennifer Widom, Data Management for XML: Research Directions, IEEE Data Engineering Bulletin 22(3), September 1999.
[9] Managing Your Data the XML Way: Data Transformation, Exchange and Integration. Oracle white paper, january 2002.
[10] Florescu D., Kossmann D. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. INRIA research report, mai 1999. RR-3680.
[11] Zhang J. Application of OODB and SGML techniques in text database : an electronic dictionary system. Sigmod records, 24(1), march 95
[12] Papakonstantinou Y., Garcia-Molina H., Widom J. Object Exchange Across Heterogeneous Information Sources. ICDE'95.
[13] Baru C. XViews: XML views of relational schemas San Diego Supercomputer Center technical report SDSC TR-1999-3. Available at : <http://www.npaci.edu/DICE/Pubs/xviews.pdf>
[14] Canfield K., Sorace J. Mapping XML Documents into Databases: A Data-Driven Framework for Bioinformatic Data Interchange. AMIA 2000: Annual Symposium on Converging Information, Technology and Health Care.
[15] Klettke M., Meyer H. XML and Object Relational Database Systems Enhancing Structural Mappings Based On Statistics. WebDB 2000.
[16] Kleiner C., Lipeck U. Automatic Generation of XML DTDs from Conceptual Database Schemas. "Web Databases" Workshop of the Annual Conference of the German and Austrian Computer Societies in Vienna, 2001.
[17] Kuikka E., Eerola A., Miettinen A., Porrasmaa J., Ek M. and Komulainen J. An object-oriented method to create an SGML DTD of an electronic patient record, University of Kuopio Research Report A/1999/6. Available at : <http://citeseer.nj.nec.com/289634.html>
[18] Bourret R. DTD Parser and Schema Converters. Available at : <http://www.rpbouret.com/schemas/index.htm>