Transitioning Existing Content: Inferring Organization-Specific Document Structures

Arijit Sengupta[1]
Sandeep Purao[1, 2]
Email: {spurao,asengupt}@gsu.edu
Phone: 404 651 3880, Fax: 404 651 3842

1: Department of CIS, Robinson College of Business, Georgia State University, Atlanta, GA 30302.
2: Institutt for Informasjonvitenskap, Agder University College, Tordenskj 65, Kristiansand, Norway

Abstract

A definition for a document type within an organization represents an organizational norm about the way the organizational actors represent products and supporting evidence of organizational processes. Generating a good organization-specific document structure is, therefore, important since it can capture a shared understanding among the organizational actors about how certain business processes should be done. Current tools that generate document type definitions focus on the underlying technology, emphasizing tags created in a single instance document. The tools, thus, fall short of capturing the shared understanding between organizational actors about how a given document *type* should be represented. We propose a method for inferring organization-specific document structures using multiple instance documents as inputs. The method consists of heuristics that can combine the individual document definitions that can be easily compiled using any of the available tools. We propose a number of heuristics utilizing artificial intelligence and natural language processing techniques. As the research progresses, the heuristics will be tested on a suite of test cases representing multiple instance documents for a few different document types. The complete methodology will be implemented as a research prototype.

1. Introduction

With the advent of the world-wide-web, a distinct shift is being felt toward the use of documents as the source of corporate data. Instead of proprietary formats, more and more businesses are exploiting the web for the purpose of their data creation, storage and distribution requirements. The introduction of XML and with it, higher control over the document structures and meta-data has clearly increased the potential of document data management on the web [Goldfarb and Prescod 2000]. However, along with the power of representation, XML introduces newer problems of structural inconsistencies. XML was designed for evolutionary data management, where authors did not need to conform to a fixed structure to validate the documents. In XML, a document can be easily parsed so long as it is *well-formed,* a constraint that allows parsers to build the document hierarchy without the use of a document type definition (DTD). However, in many cases, users do need a DTD to get started. The property of well-formedness ensures that an electronic representation of the hierarchical structure of the document can be constructed from source documents without ambiguity. A DTD can be constructed from such generated structure [Shafer, 1995; Garofalakis, 2000; Berman and Diaz, 1999]. However, such "reverse-engineered" DTDs can often include idiosyncratic decisions by the author of the XML document and may not be sufficiently generic or even too generic to be meaningfully used as a DTD. Such flaws can render the DTDs less usable.

If XML is to succeed, we must bring into fold existing documents already created by users, and reuse structures (implicit in these) for the creation of new instance documents as well as for manipulating the existing documents. The well-documented benefits of XML [Goldfarb and Prescod 2000] cannot be realized unless the large base of documents that do not yet confirm to the XML infrastructure are provided the required infrastructure. Our aim is to create tools that would like to facilitate this transition. The objective of this research, therefore, is to: *develop a methodology to semi-automatically generate high-quality, organization-specific document type definitions from the instance-specific DTDs compiled using standard algorithms.*

Our aim, thus, is not to create another algorithm to reverse engineer DTDs from individual documents. We realize that there could be documents with potentially inconsistent structures, possibly even within an organization. We propose a method for semi-automatically generating a DTD that can capture the essence of multiple documents with the same intent but possibly inconsistent structures. Such a DTD would most likely not be used for validating proposes, but for capturing the domain model to be used for creation of new documents and reorganization of existing documents. We realize that the such a process will need real or simulated intelligence, and may require direct assistance from a human operator. Our process, thus, consists of a method involving the identification of potential troublesome situations, and identifying "thumb rules" or heuristics that will enable an algorithmic approach for generating the domain model.

As an example, If we use a DTD reconstruction tool such as DDbE, it will generate a DTD that is guaranteed to parse the document(s). DTDs generated with only parseability in mind may work perfectly for parsing the existing documents, but the structure represented by such DTDs may leave much to be desired. As aptly stated by Kay [Kay, 1999????]:

> "The resulting DTD will often contain rules that are either too restrictive or too liberal. The DTD may be too restrictive if it prohibits constructs that do not appear in this document, but might legitimately appear in others. It may be too liberal if it fails to detect patterns that are inherent to the structure: for example, the order of elements within a parent element. These limitations are inherent in any attempt to infer general rules from a particular example document."

Our goal is to reduce such limitations by considering multiple documents, and if necessary, documents with the same conceptual structure, but from possibly different enterprises. The idea is to capture the core domain model, and generate the DTD that may parse a large proportion of the source documents. More importantly, however, our method aims to capture the concept common to the documents that could be used for conversion, exchange, and creation of standardized and organization-specific documents. Normally authors writing new XML documents will require something to start with, and this could give them an ideal starting point. Authors may then use XML's evolutionary document type concept to improve or change the DTD (s)he started with. One of the applications, for example, would be to automate parts of the workflow generation process with the document structures

2. Related Research

2.1 Automatic DTD Generation

A number of tools that can perform the task of extraction of the document type definition from XML documents have been built. DDbE (Data Descriptors by Example) [Berman and Diaz, 1999] is a Java library from IBM Alphaworks that can generate a DTD from an XML document. DTDGen [Kay, 1999????] is another freely available software that uses some simple rules to generate DTDs from instance documents. A recently developed system, XTRACT [Garofalakis et al, 2000] is under patenting process from Bell Labs which also has similar functionality. A somewhat more dated, but quite well-known tool, Fred [Shafer, 1995] is also capable of generating DTDs from arbitrary SGML documents (and hence can also work with most XML documents). All of these tools

aim at generating DTDs for the purpose of validating, but they fall short of generating a DTD that will be appropriate for that class of documents.

INCLUDE EXAMPLE SNIPPETS OF WHAT THESE DO. WITH PURAO's ASSIGNMENT?

2.2 Database Reverse Engineering

An analogy to the work on reverse engineering DTDs as demonstrated by the tools described above can be found in the rich stream of literature on database reverse engineering, which has focused on creating logical or conceptual data models from database instances [WCRE 1999]. A rich stream of research has focused on various reverse engineering problems such as reengineering of entities, attributes, binary relationships and ternary relationships. The core technological model for these efforts has been the relational data model. The research stream has resulted in a number of practical and innovative results. There are some key differences between reverse engineering document structures and reverse engineering databases, which make a direct application of these results.

First, documents contain highly unstructured data unlike that seen in most relational databases. Second, while a document structure can contain elements, attributes and sometimes, entities, our focus is on individual document types instead of a collection of document types arranged in a schema. The XML schema standard is still under development, and the functionality of extended links among documents is also poorly implemented, unlike database schemas, which have a well-established foundation.

In one sense, our approach mirrors that taken for database reverse engineering, which considers a number of instances to identify entities and attributes of interest in the relational or conceptual schema. The existence of multiple instance documents also sets apart our document structure reverse-engineering effort as opposed to those proposed by others (FRED, XTRACT, DDbE) which have focused on a single XML instance document. We can, therefore, use lessons from database reverse engineering for our problem but should be careful to account for the above differences.

3. Inferring Document Structures

For the purpose of this research, we assume that a simple DTD can be directly compiled from an XML document using a tool such as the ones described above (e.g. FRED). For XML documents (as opposed to SGML documents), the task of generating a DTD from a set of documents is fairly easy, since well-formedness guarantees that a DTD can always be inferred from the documents. This is not true with SGML documents since start tags or end tags (or both) could be omitted for certain elements, making it hard to find where starts and ends of tags are.

Our problem is, therefore, defined as: given a set of documents (possibly clustered by different authors or sites) but all having the same intent, to develop a strategy to generate a *super-DTD* that captures the core model covering the document set. The purpose of this research is to create a generic DTD that will capture the information in the given document instances, but individual documents may require rewriting to be validated with the DTD. The first case can be fairly simple since it can represent a simple union of all the DTDs. However, a DTD representing such a union will be mostly not-so-useful. Our problem, thus, is find a suitable DTD and its variants given a set of XML instance documents that share the same intent (as decided by the authors), but possibly with different tagsets (due to different authors), and generate a DTD that captures the core structure of all the documents and its variants. Figure 1 shows the focus of our research.

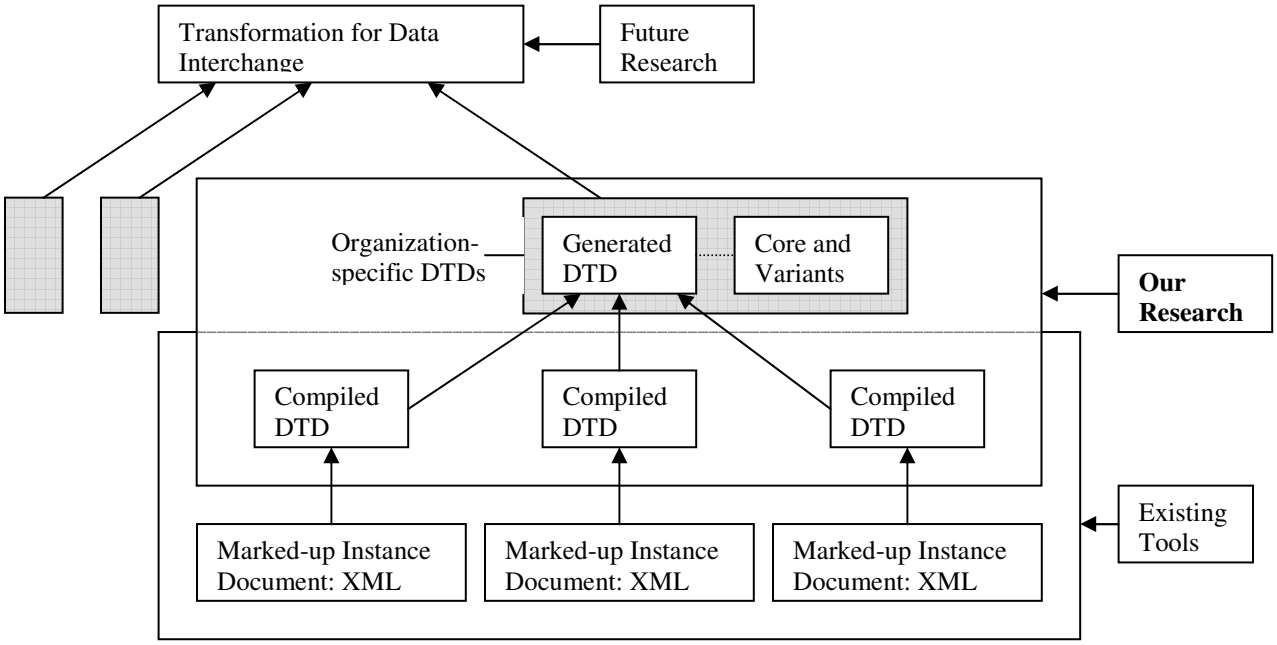

Figure 1: Research Focus

The methodology focuses on generating a DTD from multiple such DTDs, by reconciling the generated DTDs. Our approach is similar in mechanisms to other ongoing research on relational databases elsewhere [Purao et al, 2000], which proposes to develop domain models from schemas reverse-engineered from multiple relational database models. Like entities and attributes in relational model, the elements and attributes in XML can be identified in a generic manner.

4. Document Structure Generation Heuristics

A number of issues need to be considered for the generation of a normative document type definition from a set of possibly inconsistent documents with the same general structure. Although in some cases a definitive solution can be reached, most cases require the use of some intelligence and experience to decide on a resulting structure. In this section, we first describe the issues in generating a single document type from possibly inconsistent document structures, and then propose heuristics that will result in a potential solution.

a. *Elements versus attributes:* We consider elements to be structurally more important than attributes, and hence select elements as the stronger option when faced with a choice.

b. *Order of elements:* Order is important in document structure. However, a perfect solution is often difficult to achieve when faced with document instances with different order between the same set of elements. We assume a database-oriented approach where the ordering between attributes is unimportant. We try to preserve order as much as possible, but ignore order between elements when ambiguous ordering is encountered.

c. *Optional versus mandatory elements:* We consider mandatory elements to be a stronger issue, and hence try to make elements mandatory when there is a conflict.

d. *One or multiple elements:* Since one element is a special case of multiple elements, we give multiple occurrences of an element higher priority.

e. *Nesting of elements*: This is probably the most problematic issue in intelligent DTD creation. Nesting between elements indicates the presence of some form of relationship (usually "contains") between the elements. However, different document instances may nest the same pair of elements differently. A perfect solution is often hard to find in such cases though heuristics can generate satisfactory solutions.

f. *Links:* In this presentation, we do not give special attention to links. We treat them as standard

attributes.

g. *Required vs. optional attributes:* As for elements, required attributes are viewed as stronger and hence given higher priority.

Given the above structuring issues, we now define a set of heuristics that produce a resulting document structure from a set of DTDs. As a running example, we take the case of a simple book structure, with bibliographic information, header elements, the actual body text and the tail elements.

4.1 Intelligent Heuristics

1. Elevating Attributes to Elements: An attribute is promoted to an element when it appears as an *element* in one or more input DTDs and as an *attribute* in at least one other. Since we are interested in creating a normative domain model, if at least one application included this as an element, it was judged important enough to be an element and, hence is included. A variation of this heuristic could involve searching for attribute names that have common stems to element names. Then, promote the attribute to an element. Designer interaction may be necessary because there is a higher possibility of error.

   e.g., An application may have the books publication year as an element and another as an attribute. Result: we make publication year an element and retain the structure of the former document.

2. Partial or Complete Overlap between Element Names: An element name in one DTD may be fully contained in an element name from another. This suggests that, in the original designs of the document structures, the same concept was being modeled. Therefore, the element names are parsed to find the stems and from this suggestions made to the designer about the possibility of additional elements(s).

   e.g., An application may have a "heading" element under a chapter and another may explicitly call it "chapterheading". In such cases, designer is suggested to choose between the two or retain both.

3. Optional versus Required Elements: If it is indicated as required in at least one of the instance documents, include it as required otherwise optional.

   e.g., A document instance may have a required section_no element in a section, where in another instance it may be optional. Result: make the section_no required, with a default value.

4. Single versus Multiple Occurrences of an Element: If an element appears with more than one cardinality in at least one document instance, include it with higher cardinality in the DTD.

   e.g., A particular book structure may have "editor" as a multivalued element, where another structure may only allow one editor. Result: make editor a multivalued element.

5. Restructuring Elements: Ambiguous structuring of elements may cause additional problems in producing a satisfactory result. It may be possible that an element is the sub-element of one element in one structure, and of a different element in another. For example, a book may have a title, and a chapter may also have a title. A decent solution would be to include both the relationships. However, on the other hand, if a structure includes subtitles in titles, and another includes only character data, the result may be an optional subtitle in title.

6. Elements that Appear More Than Once If an element appears in more than one input DTD, then it is added to the output DTD. It is assumed that an element that appears in more than one application-specific DTD is not dependent upon organization-specific processes and, thus represents an important domain concept.

   e.g., If the book element appears in at least two documents, include it as an element.

7. Derived Attributes Remove derived, that is, computed attributes from the DTD. This functionality can belong in transformations, that is, XSL.

   e.g., If an element 'weight' is specified in grams (an attribute), and another element 'calories-per-gram' specifies the calories, the computed element, 'totalcalories' need not be included in the DTD.

5. Proposed Implementation and Testing

A prototype of the methodology is being developed. The implementation will use transformations with the help of XSLT and heuristics and natural language programming techniques will be implemented using Java™. Different versions of some of these heuristics are being tested for relational database reverse engineering [Purao et al. 2000]. These will be adapted and enhanced and new heuristics developed as needed for the proposed research. We will test the tool using content management DTDs created for managing lecture and course management content. The reconciliation will be done manually as well as with our tool in order to understand the behavior of the heuristics and how they can be improved.

References

1. L. Berman, A. Diaz. Data Descriptors by Example (DDbE). IBM Alphaworks research project documentation (http://www.alphaworks.ibm.com) June, 1999.
2. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim. XTRACT: A System for Extracting Document Type Descriptors From XML Documents. To appear in Proceedings: ACM SIGMOD'2000, Dallas, June 2000.
3. C. Goldfarb, P. Prescod. The XML Handbook. Upper Saddle River, NJ: Prentice Hall PTR, 2000.
4. S. Purao, V. Storey, M. Moore, A. Sengupta. Reconciling and Cleansing – an Approach to Domain Models. New Working Paper. February 2000.
5. K. Shafer. Creating DTDs via the GB-Engine and Fred. In Proceedings: SGML'95 Conference – Boston, MA, Dec 1995.
6. WCRE, 6th Working Conference on Reverse Engineering 6 - 8 October 1999 at Atlanta, GA.