TRANSITIONING EXISTING CONTENT: INFERRING ORGANIZATION-SPECIFIC DOCUMENTS

Arijit Sengupta[1], Sandeep Purao[1, 2]
1: Department of CIS, Robinson College of Business, Georgia State University, Atlanta, GA 30302. Tel: +1 (404)651-3880 Fax: +1 (404)651-3842 E-mail: {asengupt,spurao}@gsu.edu, URL: http://www.cis.gsu.edu
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 performed. 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 combine individual document definitions, which may have been compiled using standard algorithms. 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 different document types. The complete methodology will be implemented as a research prototype.

KEYWORDS: Document Management, Document Models, Reverse Engineering, Heuristics, Document Type Definition

INTRODUCTION

Documents are a central part of every organizational process. They are used to capture information for archival, publication or as intermediate records of organizational processes. 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 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). Along with the power of representation, however, XML has introduced newer problems of structural inconsistencies. XML was designed for evolutionary data management, where authors did not need to conform to a fixed structure to create valid 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 requiring a pre-defined document type. 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. However, in many cases, it is necessary to capture and store the metadata as a document type definition (DTD) or a schema. A DTD can be constructed, using standard algorithms, from the structure evident in a marked-up instance document (Shafer, 1995; Garofalakis, 2000; Berman and Diaz, 1999; Kay, 2000). Such reverse-engineered DTDs, however, simply end up capturing decisions about markup by the author of the XML instance document. The markups may not be sufficiently generic, may enforce incorrect cardinality, may even be too generic, or may use inappropriate nesting. All these can significantly reduce the usefulness of the generated DTD, rendering them less usable.

If XML is to succeed, we must bring into fold existing documents and reconcile, where necessary, implicit assumptions underlying their content and structure. The well-documented benefits of XML (Goldfarb and Prescod 2000) cannot be realized unless the large base of documents that do not yet conform to the XML infrastructure are provided the required infrastructure. Ensuring that a class of documents conforms to a common, shared intent and structure, therefore, will require a reconciliation of the various, probably different, declarations of tags, metadata and structure that different organizational actors may assign to comparable parts of different instance documents. For example, consider a memo that contains information about a customer complaint and its resolution. The parts of such a memo, disregarding the headers, may be variously declared as <text (complaint, resolution)> or <body (complaint, complainttype, resolved, comments)> or <response (complaint, action)> or something else. Inferring organization-wide document structures that may be shared by these actors should, then, improve and streamline the related processes. Further, the benefits of aggregating the information available in these instance documents should be clear. Resolution of different structures and tags that different actors may impose on individual instance documents is, therefore, a prerequisite for this reconciliation.

Our goals in this research are to create techniques and tools that can facilitate this process of reconciliation. 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.* A DTD reconstruction tool such as DDbE will generate a DTD that is guaranteed to parse the source document(s). Our intent, instead, is to capture the core document model, and generate a DTD that will parse a large proportion of the source documents, and could be used for conversion, exchange, and creation of standardized and organization-specific documents.

RELATED RESEARCH

Document Management

Documents can capture meanings and purposes utilized and produced by organizational actors, by exploiting technological features necessary for the storage and processing of that information (Tyrväinene and Päivärinta, 1999). Electronic document management is the capture, management and manipulation of documents in an organized manner. A rich stream of research exists on document management (Sprague, 1995). A document, among other things, can be considered as recorder of contents, evidence of action or proof of authenticity. (Tyrväinene and Päivärinta, 1999), in fact, describe as many as eleven different facets of documents, which they identify based on an analysis of writings in IS research. Markup languages such as SGML and XML are technologies, which, along with others, can make electronic document management easier and more effective. Standardization of documents, whether using these technologies or otherwise, is an important aspect of document management. Several ongoing research efforts (see, e.g. Jyväskylä, 2000) emphasize the importance of document standardization and report on the high costs in terms of time, money and effort that organizations incur when they embark on document standardization projects. Standardization of documents may involve the standardization of usage or content of documents. The first requires an understanding of and conformance to organizational processes as well as buy-in from organizational actors. In contrast, standardization of the second involves analysis and reconciliation of varying perspectives that are made explicit by the organizational actors. Our focus in this research is on the latter. Current research suggests that when performed manually, this kind of standardization can be a long and arduous process, taking much effort and involving much expense. In the context of XML technology, this translates to standardization of the document type definition (DTD) or schema. Since the schema standard (XSchema, 2000) is still fairly new, our focus in this paper is on the standardization of organizational DTDs.

Automatic DTD Generation

A number of approaches, research prototypes and state-or-the-art commercial tools can extract (reverse-engineer) the document type definition from XML instance documents. DDbE (Data Descriptors by Example) (Berman/Diaz, 1999) is a Java library from IBM Alphaworks that can generate a DTD from an XML document. A recently developed system, XTRACT (Garofalakis et al, 2000) is under patenting process from Bell Labs which also has similar functionality, but can generate more optimized DTDs for single productions. 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). A web-based tool called DTD Generator for this purpose using the Saxon XML parser is also available (Kay, 2000). All of these tools aim at generating DTDs for the purpose of validating. The techniques they use include different ways of 'guessing' the intended structure from different combinations of the same set of tags (Berman/Diaz, 1999; Garofalakis et al, 2000). Some of these tools also try to distinguish the special attributes such as links, IDs, IDREFs, and so on, using heuristics based on the number of times an attribute value is used (Kay, 2000).

The tools described above excel at constructing a DTD for a given instance document. They can infer cardinalities, nesting of elements, default values of attributes, as well as references (links) between elements. The tools, however, clearly fall short of generating a standardized DTD that may be appropriate for that class of documents. Their focus, instead, is on the compilation of a DTD that can make explicit the content and structure of the XML instance documents. Our research, then, involves use of these DTDs as inputs to infer a standardized DTD for the organization. The existence of multiple instance documents sets apart our document structure reverse-engineering effort as opposed to those proposed by others (FRED, XTRACT, DDbE, DTD Generator) which have focused on a single XML instance document. In one sense, our approach mirrors that taken for database reverse engineering, where a number of instances are considered to identify entities and attributes of interest in the relational or conceptual schema. We can, therefore, use and extend lessons from database reverse engineering for our problem but should be careful to account for any differences.

Database Reverse Engineering

A rich stream of research on database reverse engineering has focused on creating logical or conceptual data models from database instances (WCRE 1999). Various problems have been addressed and solved to varying degrees such as reverse engineering of entities, attributes, binary relationships and ternary relationships (Chiang et al., 1994, Permerlani and Blaha 1994). The core technological model for these efforts has been the relational data model. The research has resulted in a number of practical and innovative results (Fahrner and Vossen, 1995, Chiang 1995). Though it is possible to extend and augment these approaches for our context –organizational document documents – there are sufficient differences between 'database reverse engineering' and 'organizational document standards inference' to warrant a different approach.

First, documents can contain highly unstructured data unlike that seen in most relational databases, making inferences based on data instances difficult. Second, unlike relational database schemas, which have a well-established foundation, the XML schema standard (XSchema 2000) is still fairly new (May 2000), and the functionality of extended links among documents is also poorly implemented. The focus of our efforts, therefore, is on individual document types without the benefit of exploiting information that may be available in links across documents. Third, perhaps most important, is the nature of data instances used as inputs. For reverse engineering of relational database schema, the inputs contain multiple database designs, which are generated from the data by *one* source, say, a program written with a second generation language. They share a common layout but do not have any metadata associated with them. The reverse-engineering process is, then, expected to discover this metadata and its structure. On the other hand, the document instances are marked up by different sources (*e.g.*, different authors) and can contain different tags indicating the same or different metadata, arranged in different layouts. The problem of inferring a standardized document type, thus, involves more than a mere extension of database reverse engineering approaches.

Our research question and focus – inferring organizational documents – is, therefore, shaped by three research streams: document management, automatic document type generation and database reverse engineering. We motivate it from a document management perspective. The automatic DTD generation provides appropriate inputs to our process. Finally, the database reverse engineering research suggests approaches that we adapt and extend for document type generation.

INFERRING ORGANIZATIONAL DOCUMENT STRUCTURES

A simple DTD can be directly compiled from an XML document using a tool such as the ones described above (e.g. DDbE). For XML (as opposed to SGML), the task of generating a DTD from a document is not very hard, since well-formedness guarantees that the document structure (the tag hierarchy) can always be inferred from the documents. Our problem is, therefore, defined as: given a set of documents, possibly clustered by different authors or sites, but all having the same intent, and their associated DTDs, compiled with standard algorithms, to develop a strategy to generate a *super-DTD* that captures the core model covering the document set. One fairly simple approach to this would create a simple union of all the DTDs. However, a DTD representing such a union will be mostly redundant, and even a union would require choices to reconcile mismatches in sequencing and nesting. Our problem, thus, is to find a suitable DTD and its variants given a set of XML instance documents that share the same intent but have different tagsets (due to different authors). The generated DTD should capture the core structure of all the documents and its variants. Figure 1 below shows the focus of our research.

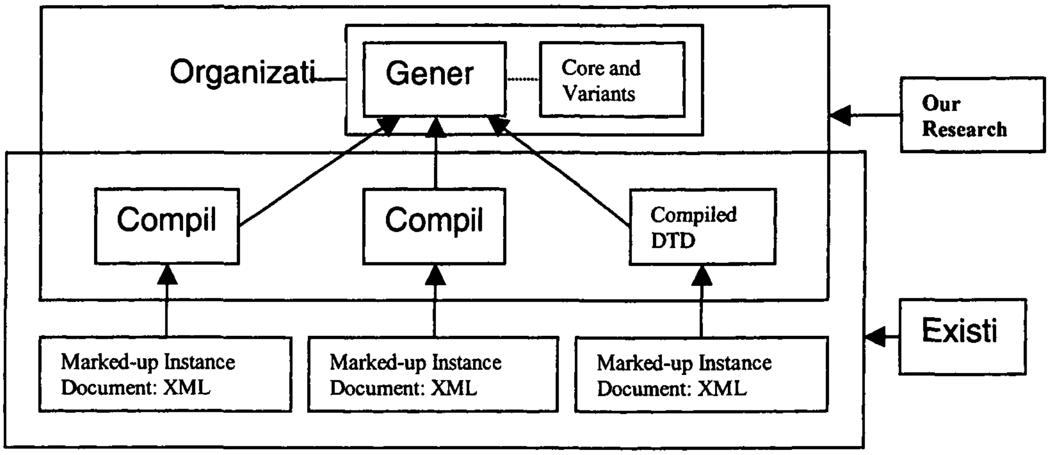
Organizati
Gener
Core and Variants
Our Research
Compil
Compil
Compiled DTD
Existi
Marked-up Instance Document: XML
Marked-up Instance Document: XML
Marked-up Instance Document: XML

Figure 1: Research Focus

We use as inputs the DTDs compiled from instance documents by existing tools. These DTDs contain elements, attributes, their sequences, cardinalities and structures. Our approach, then, is focused on creating a core DTD plus its variants, which, taken together will cover a substantial proportion of document instances of this document type. The core DTD and the variants may be combined with use of ENTITY's, that is, with the help of mechanisms resembling aggregations or inheritance, ensuring coverage of a very large proportion of source instance documents.

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, propose heuristics that will result in a potential solution, and show an application.

Elements versus attributes: Elements contain and envelop parts of a document. On the other hand, attributes provide slots for further information about these parts. In a choice between elements and attributes, therefore, it may be argued that the former should have primacy. There may be exceptions to this rule, for example, if rendering of values depends upon whether a certain piece of information is an element or an attribute.

Order of elements: Unlike a relation in a relational database, ordering between elements is important in a document. When document instances contain different sequences, but similar sets of elements, heuristics may generate a compromise, but not a perfect solution. On the other hand, ordering between attributes is unimportant (W3C 1998).

Cardinality of elements: Two issues are important here. The first concerns whether the cardinality of an element is optional or mandatory. If an element is marked as mandatory in one instance document and optional in another, there are clearly two approaches to resolve this mismatch. One is to enforce the mandatory cardinality, forcing instance documents conform to this requirement. This can mean a strict interpretation, giving primacy to a more rigid set of definitions. The other is to allow an optional cardinality, covering a larger proportion of instance documents with the definition. Additional information may be necessary to devise a heuristic that resolves this anomaly. Another aspect of cardinality is the choice between one or multiple occurrences if a mismatch is found between instance documents. This can represent a choice similar to optional versus mandatory, though single occurrence may be considered a special case of multiple elements, giving multiple occurrences higher priority during reconciliation.

Required vs. optional attributes: A parallel to the choices above can also be seen in the choice for required or optional attributes. This decision too, may be dictated by concerns similar to the ones described above. In addition, manipulations and renderings applied to the attributes may be examined to decide the appropriate reconciliation.

Nesting of elements: This is perhaps 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. It is conceivable that the nesting may be reversed or may contain different levels. A set of heuristics that combines parsing as well as tree traversal may be needed for meaningful reconciliation of any mismatched nesting structures between document instances.

Links: Given the focus of the research – a document type, instead of schemas containing different document types – we treat links as standard attributes. Since the linking standard is still evolving (XLink 2000), this aspect of reconciliation can be retained for the next phase of our research project. To address the above issues, we now define a set of heuristics that can produce a resulting document structure from a set of DTDs.

Intelligent Heuristics

The heuristics are grouped in three categories: element reconciliation, attribute reconciliation and structure reconciliation. We describe these below with the help of simple examples.

Element Reconciliation Heuristics

1. **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.
2. **Optional versus Required Elements:** If it is indicated as required in at least one of the instance documents, include it as required, with a default value.
   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.
3. **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.
4. **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 DTD, it is not dependent upon naming idiosyncrasies and, thus represents an important domain concept.
   e.g., If the book element appears in at least two documents, include it as an element.

Attribute Reconciliation Heuristics

5. **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 document model, if at least one document includes this as an element, we reason that it was judged important enough to be an element and include it. A variation of this heuristic could involve searching for attribute names that have common stems to element names, and promoting these attributes to elements. Designer interaction may be necessary here because of a higher possibility of error.
   e.g., A document may have publication year as an element and another as an attribute. Result: make publication year an element and retain the structure of the former document.
6. **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.

```
<!ELEMENT BOOK (TITLE, AUTHOR+, EDITOR, PUBLISHER, PRICE, BODY)>
<!ATTLIST BOOK PUBYR CDATA #IMPLIED>
<!ELEMENT TITLE (#PCDATA | SUBTITLE)*>
<!ELEMENT BODY (CHAPTER+, APPENDIX+, REFERENCES)>
<!ELEMENT CHAPTER (CHNO, HEADING, CHBODY)>
<!ELEMENT CHBODY (#PCDATA | SECTION)*>
<!ELEMENT SECTION (SECNO, HEADING, SECBODY)>
```

DTD 1. A partial Book DTD instance

```
<!ELEMENT BOOK (TITLE, AUTHOR+, EDITOR+, PUBLISHER, PUBYR, PRICE,
     PRICEINDOLLARS, BODY)>
<!ELEMENT BODY (CHAPTER+, APPENDIX+, REFERENCES)>
<!ELEMENT CHAPTER (CHNO?, CHHEADING, CHBODY)>
<!ELEMENT CHBODY (#PCDATA | SECTION)*>
<!ELEMENT SECTION (SECNO?, SECHEADING, SECBODY)>
```

DTD 2. Another Partial Book DTD instance

Figure 2 Input DTDs

Structure Reconciliation Heuristics

7. **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 reasonable solution would be to include both the relationships. In contrast, if a structure includes subtitles in titles, and another includes only character data, the result may be an optional subtitle in title.

The heuristics we have outlined above represent a start towards an approach for reconciling the input DTDs to create a super-DTD. Additional heuristics that are in the process of development include the identification of core DTD, and the use of entities to combine modules to create variants. As the XSchema standard evolves (XSchema 2000), we expect that the parsing techniques in our heuristics will also exploit information about data types in the source documents, generating a Schema from a group of source DTDs. The above set of heuristics, however, can be used to generate a plausible super-DTD. We demonstrate below how these heuristics may be applied with an example.

```
<!ELEMENT BOOK (TITLE, AUTHOR+, EDITOR+, PUBLISHER, PUBYR, PRICE, BODY)>
<!ELEMENT TITLE (#PCDATA | SUBTITLE)*>
<!ELEMENT BODY (CHAPTER+, APPENDIX+, REFERENCES)>
<!ELEMENT CHAPTER (CHNO, CHHEADING, CHBODY)>
<!ELEMENT CHBODY (#PCDATA | SECTION)*>
<!ELEMENT SECTION (SECNO, SECHEADING, SECBODY)>
```

DTD 3. DTD Generated from application of heuristics.

Figure 3: Reconciled Super DTD

APPLICATION

A simple instance of the above heuristics is shown in Figures 2 and 3. Figure 2 shows the input DTDs, compiled from source documents, marked up by different authors, in our case, different publishers. These DTDs may have been compiled using any of the tools mentioned earlier. They contain specification of elements, attributes, cardinalities, sequencing and nesting. The DTDs shown are partial, that is, they show only one of the branches of the tree fully. We describe below how the heuristics may be applied to reconcile these source DTDs.

In this example, the attribute ‘pubyear’ appearing in DTD 1 can be promoted to an element using *heuristic 5*. The element ‘heading’ under ‘section’ is found to be similar to the ‘secheading’ element, and the two have been combined using *heuristic 1*. Using the same heuristic, ‘chheading’ has been retained in favor of ‘heading’ under ‘chapter.’ Using *heuristic 2*, ‘chapno’ and ‘secno’ have been made required since one of the DTDs required them. The 'editor' element has been elevated to a multiple-occurrence element using *heuristic 3*. Elements that have been included in the output DTD are elements that were included in both the DTDs (*heuristic 4*). Heuristics 6 and 7 were not applied in this example. Notice that since these DTDs were generated by reverse-engineering sample XML documents from one organization, most element names were same or similar. If the element names are different but with the same meaning, various linguistic analyses need to be performed in order to determine their similarity.

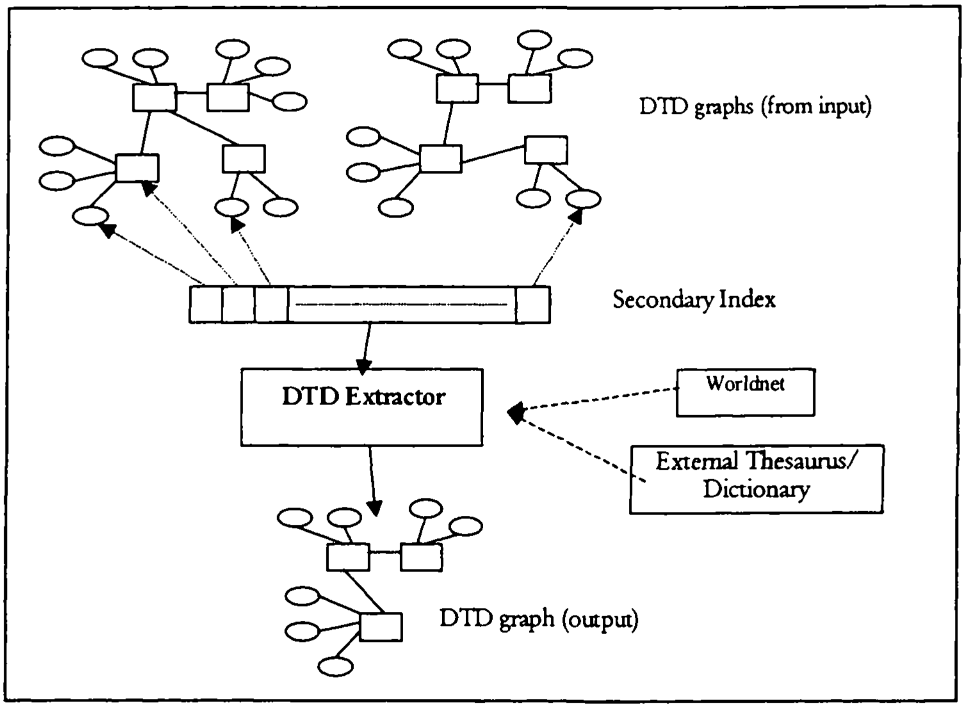

Figure 4. Architecture of DTD Extractor implementation

PROPOSED IMPLEMENTATION AND TESTING

We are developing a proof-of-concept prototype for the approach. The implementation is primarily in Java with the XP XML Parser library from James Clark. XP allows parsing of XML 1.0 compliant documents including XML DTDs, and includes a class library that can be instantiated to process different parts of the DTD. The input to this program consists of two or more DTDs (possibly with sample XML content, although the XML content is currently ignored). The parsed DTDs are used to create memory-based graph structures representing the

DTDs. The heuristics in this paper are then applied to the memory structures. The approach is similar in spirit to other ongoing research on relational databases (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. The proposed architecture of the structures and processing is shown in Figure 4 below.

Figure 4 shows the basic architecture of the system under implementation. Initially, DDbE is used to generate DTDs from input documents (although XTRACT is the most optimized DTD generator known, a public implementation is not yet available). These DTDs are read in memory-based graph-like structures, and secondary indices (currently a variant of B-tree) are created on top of the graph elements. These structures are then processed using the proposed heuristics to generate another graph structure for the output DTD, which is then used to generate the final DTD for output. Planned extensions to the architecture include hooks for complex processing such as linguistic analysis and the use of external dictionaries. We will test the tool to reconcile 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.

CONCLUSION

Good document type definitions are invaluable for maintaining consistency between organizational documents. Documents created in XML may not have a DTD, in which case one can be generated from document instances using DTD generation tools. However, these tools fall short in capturing the organizational document design, but simply capture the idiosyncratic naming decisions of individual authors. This paper describes a method for generating organizational document type definitions from multiple XML instances, possibly created from different sources. Our contribution is in using the current methods to assist in the generation of organization-specific document models. Further research, including development of a research prototype as well as experimentation is under way to assess the viability of the approach, its utility, and future applications.[17]

REFERENCES

Berman, L., A. Diaz (1999) Data Descriptors by Example (DDbE). **IBM Alphaworks Research Project Documentation** (At http://www.alphaworks.ibm.com) June, 1999.

Chiang, R., (1995) A Knowledge-Based System for Performing Reverse Engineering of Relational Databases, **Decision Support Systems**, 13, pp. 295-312.

Chiang, R., T. M. Barron, V. C. Storey. (1993) Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database, **Data & Knowledge Engineering** 12, pp. 107-142.

Fahrner, C. , G. Vossen (1995) A Survey of Database Design Transformations Based on the Entity-Relationship Model, **Data & Knowledge Engineering.** 15, pp. 213-250.

Garofalakis, M., A. Gionis, R. Rastogi, S. Seshadri, K. Shim, (2000) XTRACT: A System for Extracting Document Type Descriptors From XML Documents. **Proceedings of ACM SIGMOD'2000**, Dallas, June 2000.

Goldfarb, C., P. Prescod (2000) **The XML Handbook**. Upper Saddle River, NJ: Prentice Hall PTR, 2000.

Jyväskylä (2000) **Document Management Research Program** at University of Jyväskylä. At http://www.cs.jyu.fi/~airi/docman.html

Kay, M. (2000) Saxon DTDGenerator – A Tool to Generate XML DTDs. **Technical software Documentation**, At http://users.iclway.co.uk/mhkay/saxon/dtdgen.html. February 2000.

Premerlani, W. J., M. R. Blaha. (1994) An Approach for Reverse Engineering of Relational Databases, **Communications of the ACM**. 37, 5, pp. 42-49.

Purao, S., V. C. Storey, M. Moore, A. Sengupta (2000) Reconciling and Cleansing – an Approach to Domain Models. Working Paper, **Georgia State University**. February 2000.

Shafer, K. (1995) Creating DTDs via the GB-Engine and Fred. **Proceedings: SGML'95 Conference.** Boston, MA, December 1995.

Sprague, R. H. (1995) Electronic Document Management: Challenges and Opportunities for Information Systems Managers. **MIS Quarterly**. 19, 1. Pp. 29-50.

Tyrväinene, P., and T. Päivärinta (1999) On Rethinking Organizational Document Genres for Electronic Document Management. **Proceedings: HICSS-32**, 1999. Hawaii.

W3C (1998) **XML Specification 1.0**. February 1998.- http://www.w3.org/TR/1998/REC-xml-19980210.

WCRE (1999) 6th **Working Conference on Reverse Engineering**. 6 - 8 October 1999 at Atlanta, GA.

XLink (2000) **The Linking Specification for XML.** http://www.w3.org/XML/Linking.html

XSchema (2000) **XML Schema Specification**. At http://www.w3.org/TR/xmlschema-0/ Draft 7 April 2000.

ACKNOWLEDGEMENTS

We wish to acknowledge comments from anonymous reviewers, which have helped to considerably improve the content and the presentation. The paper has also benefited from discussions with Airi Salminen and Pasi Tyrväinene.

[17] A preliminary version of this paper, under a different title, was presented at the conference XML Meets Business in Heidelberg, Germany in May 2000.