BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **6**, No 1

Sofia • 2006

A System for Storing, Retrieving, Organizing and Managing Web Services Metadata Using Relational Database[*]

Ivo Marinchev

Institute of Information Technologies, 1113 Sofia

Abstract: *In this paper we present our system for efficient storage, update, and retrieval of web service metadata documents in relational database. Initially we investigate all of the existing approaches for efficient storage and retrieval of XML documents in relational database. As a result we selected a recently proposed structure-centered storage schema named DLN (Dynamic Level Numbering). As there is no freely available implementation of DLN method, as in its original form it does not support namespaces, and as the web services metadata documents rely heavily on namespaces we created our own clean-room implementation that we extended to support XML namespaces.*

Keywords: *XML, XML namespaces, XML Schema, Structure-Centered XML Storage, WSDL, Relational Database Management Systems, Business to Business Integration.*

Introduction

In the past several years more and more data is presented in XML files. All recent technologies rely on XML formats. In the field of the web services and especially semantic web services the usage of XML is almost ubiquitous (except WSMO [16] related specifications that prefer to use human oriented file formats, nevertheless they also have XML encoding schemes). At the same time the native XML databases are not so widely used (and applicable) to small and medium size projects (including research ones). The reason behind this fact is that native XML databases tend to be too expensive and resource hungry (requiring separated servers, clusters etc.), if proprietary, or incomplete (unstable, unreliable) in case of the several open source ones. At the same

[*] The research has been partially supported by "Technologies of the Information Society for Knowledge Processing and Management" – IIT–BAS Research Project No 010061.

time the usage of relational database management systems (RDBMS) is ubiquitous (every organization has at least one RDBMS). Contemporary RDBMS are feature complete, heavily optimized, and include enterprise features such as clustering, transactions, replications, partitioning, monitoring tools, etc. There are a lot of different flavors RDBMS for any conceivable deployment scenario – from small embedded databases such as SQL Lite and HSQL, through full featured light-weighted SQL servers as MySQL,, PostgreSQL, Firebird (Interbase) to the “heavy artillery” represented by Oracle, Sybase, Informix, Progress, SAP DB, Microsoft SQL Server, etc.

There are two types of XML databases – Native XML databases and XML enabled relational database management systems.

Native XML databases – they are built from the ground up with the purpose to manage XML data. They store the data in their internal format that is optimized for XML processing. These databases have good support for XML standards but tend to be very expensive (if proprietary). At the same time the open source XML databases tend to be non-scalable and sometimes even reliable as they are early versions (or research projects) and concentrate on feature implementations rather than rigorous testing and performance.

XML enabled relational database management systems – they are actually relational databases that are extended to output (and sometimes to input) data in XML format. In fact all major relational database vendors have added XML support to their products. Currently, two main kinds of mappings for integrating XML data into the relational model are supported [1, 2]. The first is a document-centered one, the documents are stored as a whole entity. This approach poses restrictions on query ability and intra-document updates. The second is a data-centered one and requires a XML DTD or schema to map XML data to application-specific tables and attributes. While this approach supports application specific SQL operations over the data it loses information such as element sibling order and leads to an expensive reconstruction of document fragments.

Both mappings are not able to efficiently manage large-scale document-centric XML data that must be updated and accessed via standard XML interfaces like XPath, XQuery or DOM. The later requirements can be met by a third kind of mapping named *structure-centered storage*. It does not require a XML DTD or schema and maps the tree or graph structure of XML documents generically into predefined relations. Such a generic XML storage is especially useful in data integration systems to manage highly diverse XML documents.

Classical approaches

Different relational mappings for generic storage of XML documents are proposed. In [3] the tree-like node structure of XML documents is represented as parent-child relationships, but this approach is inefficient for reconstructing documents; [4] encodes the document tree in binary relations, but also has performance difficulties for reconstructing document fragments. A multidimensional mapping using document id, value and path surrogate is published in [5]. However this approach does not deal with update operations and its path coding restricts the number of child elements per node.

A key approach to improve query and retrieval performance is the use of semantically meaningful node-ids when mapping XML data into nodes according to the Document Object Model (DOM). Therefore several numbering schemes have been proposed. One of the first numbering schemes supporting ancestor-descendant relationships was published in [6]. It labeled each tree node with a pair of preorder and postorder position numbers. So for each pair of nodes x and y, x is an ancestor of y if and only if preorder(x)<preorder(y) and postorder(x)>postorder(y). A similar scheme was chosen in [7]. While this numbering scheme is easy to compute and can be used for streamed XML data it is highly inefficient when new nodes are inserted. Here each node in preorder traversal coming after the inserted node has to be updated.

The update problem has been addressed by the *extended preorder* numbering scheme introduced in [8] and adopted in [9] as *durable node numbers*. They also use a pair of numbers for each node. The first number captures the total order of the nodes within the document like the preorder traversal but leaves an interval between the values of two consecutive nodes. The second number is a range value. As with the preceding scheme the ancestor-descendant relationship between node x and y can be determined from x is an ancestor of y iff order(x) < order(y) ≤ order(x) + range(x). With the sparse numbering insert operations will not necessarily trigger renumbering of following nodes if the difference of the order value of preceding and following node is larger than the number of inserted nodes. However inserting new sub trees with a substantial number of nodes requires renumbering as well.

In [1] a numbering scheme called SImple Continued Fraction (SICF) is proposed. It numbers the nodes from left to right and top-down. Each node number can be expressed as a sequence of integer values – adding an integer per tree level – or a fraction. This approach reduces the update scope (the set of nodes whose numbers (potentially) have to be updated) after a node insertion to the right siblings and their descendants of the inserted node. However this still can be a large number. Furthermore SICF fails if a certain tree depth is reached.

Another approach with left to right and top-down numbering is published in [10]. The so-called Unique element IDentifiers (UID) are based on a tree with a fixed fan-out of k. If a node has less than k children virtual nodes are inserted. The UID allows the computation of the parent node id and the id of child i. This approach has two main drawbacks: 1) fixed fan-out is problematic with irregular structured documents, 2) node insertion requires updates of all right siblings and their descendants.

Some of the UID drawbacks were tackled in [11] with the definition of recursive UIDs (rUID). Here the tree is partitioned in local areas allowing different fan-outs and reducing updates after insertion of nodes. However it needs access to the whole tree in order to compute the identifiers which prevents the streaming of data to be inserted.

Theoretical findings for labeling dynamic XML trees are given in [12]. The described schemes determine labels that are persistent during document updates and contain ancestor information. Furthermore lower bounds for the maximum label length are presented. However no sibling order of the XML nodes is maintained and therefore it is not suitable for general XML document management.

A recent paper proposed the so-called ORDPATH numbering scheme [13]. It is based on a hierarchical labeling scheme using a prefix-free encoding and supports insertion of new nodes without re-labeling existing nodes.

Decimal classification and dynamic label numbering methods

Since simple preorder numbering has afore-mentioned drawbacks for insert operations, order encoding based on decimal classification (DC) was introduced in [14]. DC ids are composed of a sequence of numeric values separated by a dot. The root node is assigned a single numeric value. Child node ids start with the id of the parent node appended by a dot and a numeric value, which we call the *level value*. The level value of a left sibling from a node *A* must be less than the level value of *A*. As illustrated in Fig. 1, this approach restricts the update scope after a node insertion to the right sibling nodes and their descendants. Besides this property the encoding has further advantages: 1) the parent can be computed from the id, 2) the ancestor-descendant relationship can also be determined using only the id value and 3) the ids can be sequentially assigned.

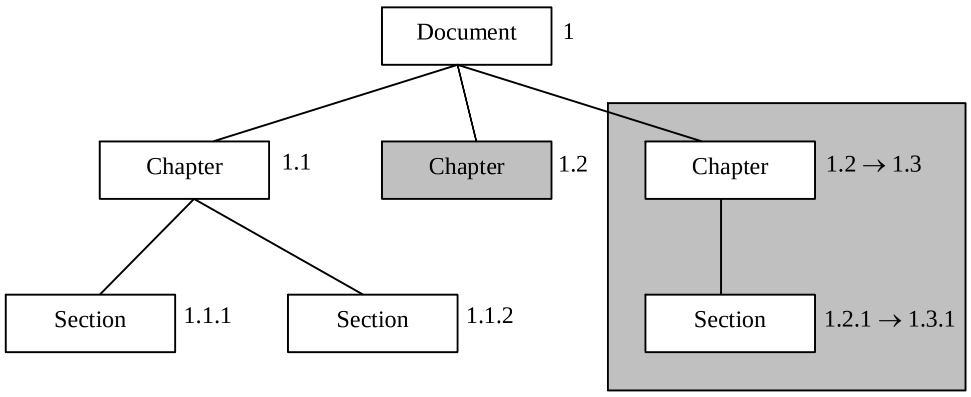

Fig.1. DC order encoding. Highlighted is the update scope after insertion of middle chapter element

Unfortunately the DC encoding also has some shortcomings. 1) the id length depends on the tree depth, 2) a binary or string comparison of the ids may deliver wrong results with respect to the total node order, e.g. comparing 1.9 and 1.10 and 3) inserting a node as a child of a parent with a high fan-out may still result in a large number of nodes to be updated.

In [14] two solutions for the second shortcoming were proposed. The simple approach uses a fixed number of digits for each level number. Thus our example may be translated to 0001.0009 and 0001.0010 which now delivers a correct result using string compare. However this solution restricts the maximum fan-out of a node to a fixed value and on the other hand uses too much storage space for child nodes with few siblings resulting in long path ids. Therefore the second approach uses UTF-8 (Unicode Transformation Format-8, defined in RFC 2279) encoding for the level numbers where small values can be represented by single bytes and larger values by two or more bytes. This encoding results in smaller path ids and permits binary compare operations.

With the UTF-8 encoding we still need at least one byte per level value. Given that most nodes of a XML document have a low fan-out this seems to be a waste of storage space. As a consequence of this and the still unsatisfying update behavior in

[15] proposes a new numbering scheme called Dynamic Level Numbering (DLN) which is based on DC.

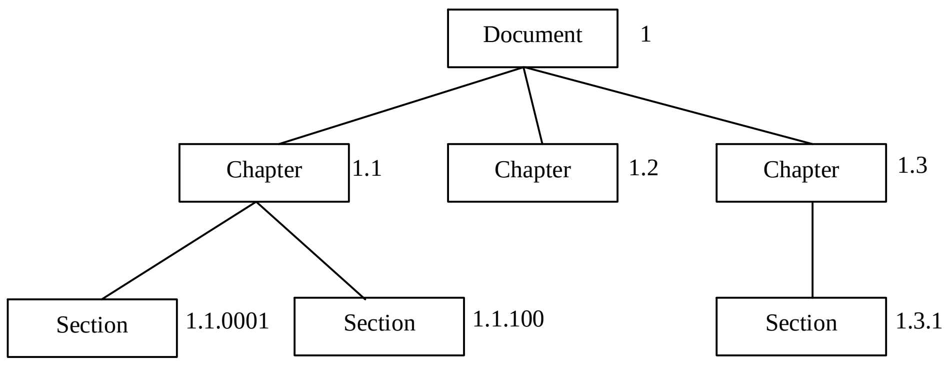

Fig. 2. DNL numbering with adjusted number of digits

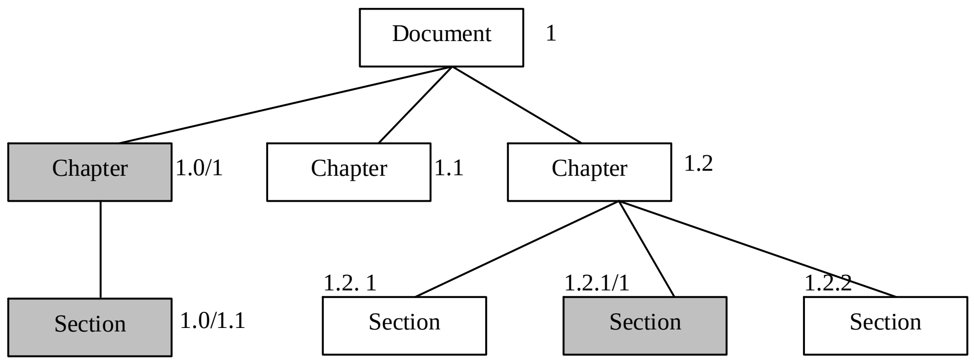

Fig. 3. IDS after insertion of left chapter subtree and middle section node in chapter 1.2

DLN contains solutions for the stated problems two and three of DC encoding and has an efficient binary representation, which tackles problem one. In order to obtain comparable ids a fixed number of digits for level values (fixed length) is used. This restriction can be relaxed so that only the level values of sibling nodes need to have the same lengths. Hence the number of digits per level value can be dynamically adjusted according to the number of sibling nodes. In the example of Fig. 2, the siblings at the second level use 1 digit, while the descendants of node 1.1 at the third level use 3 digits per level value.

DLN reduces the renumbering effort after insert operations by the introduction of *subvalues*. The basic idea is that between two consecutive level values *a* and *b* one can add further values by adding a suffix to *a*. The resulting ids need to be larger than all ids of children nodes of *a*. This is accomplished by inserting a special character between *a* and the suffix which is greater than the dot separating the level values.

The application of subvalues is shown in Fig. 3. Nodes 1.0/1 and 1.2/1 could be inserted without renumbering the existing nodes. It is important to note that the inserted chapter node must not get level value 0 (or id 1.0). Otherwise we would have no possibility to add further nodes via subvalues to the left of it. Subvalues can be used recursively. For instance to insert a node between nodes with ids 1.1/1 and 1.1/2 we can add a further subvalue level and assign 1.1/1/1 to the new node. The only disadvantage of subvalues is the increased id length.

There is also a binary representation of DLN ids. The level values are binary coded using the same number of binary digits for all sibling nodes as required before. To separate the level values and to distinguish between values of the next level and subvalues only one bit. '0' is used that means the following value belongs to the next hierarchy level whereas '1' depicts a subvalue.

Before we discuss the properties of our encoding we give some examples to demonstrate the transformation into binary representation. If we use 2 bits for the first level and 4 bits for the second level we would encode our examples 1.09 as 01 0 1001 and 1.10 as 01 0 1010. To insert a node between both siblings without renumbering we have to use a subvalue resulting in 1.09/01 or in binary notation 01 0 1001 1 0001.The length of a subvalue should be identical to the length of the level value. With this property we can minimize the metadata needed to calculate the ids of following nodes and the parent node id.

Implementation and applications

The DLN approach is suitable for our needs as it supports a wide range of XML documents, especially:

- irregular documents having nodes with low fan-out as well as high fan-out. All WSDL, XML Schema are samples for irregular documents;
- large documents which can only be sequentially inserted;
- explicitly express the total nodes order to allow node clustering for high retrieval performance of document fragments using sequential scans;
- reduce the necessity for renumbering after insert operations;
- assist in the efficient processing of XPath queries, e.g. containment queries;
- in order to use this numbering scheme with a conventional RDBMS it should be exploitable by the indices and query optimizer of the database system.

In our implementation we do not currently employ binary representation of DLN ids. We use string representations for easier debugging and testing of more complex algorithms needed for XPath support. String values bring, of cause, some performance degradation but corresponding code is abstracted in separated classes and can be easily changed to binary representation in the final version.

Fig. 4 shows the database ER diagram of the database used by our system. It contains the following tables. The foreign keys that correspond to relationships are implied. Form this conceptual model we generate concrete database schemas corresponding to different possible databases. In practice we tested our system with MySQL and MS SQL Server, but it should work with any SQL compliant RDBMS as only basic SQL features are employed.

The rest of our system is implemented as Java library that uses SAX (or DOM) parser for processing raw XML documents. We are considering now to use stream based parser (for example StaX) that more naturally fits in our settings.

We have built our system with the intention to organize a large collection categorized web services [17] gathered from salcentral [18] and Xmethods [19].

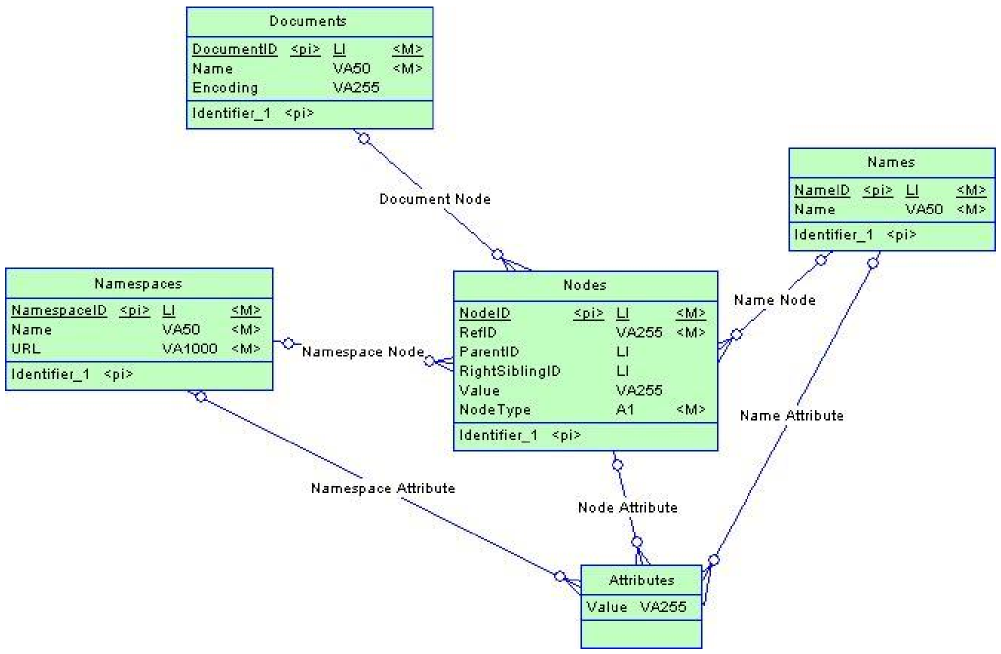

Fig. 4. Conceptual data model used in our system

In the original data the services are classified as a set of file-system directories and subdirectories. A TXT file with the following structure accompanies each WSDL file:

1 line = service provider

2 line = original classification by SALCentral

3 line = service name

4 line = URL of the original WSDL file on the Web

5 line = Plain text description of the service crawled from the SALCentral/ Xmethods web page

We took a different approach in order to create a coherent representation of the web service metadata contained in this representation. Actually we represent all of the data as XML files. The directory structure as a big XML file and the TXT descriptions of the services as an associated XML files:

```
<service_description>
  <service_id></service_id>
  <service_provider></ service_provider>
  <original_classification></original_classification>
  <service_name></service_name>
  <url></utl>
  <description></description>
</service_description>
```

This way we can store all of the data in our system in an uniform way and then use simple XPath queries to retrieve any part of it. Such representation is much more convenient for processing and analysis than the original bunch of directories. Our unified approach shows its advantages when the annotated OWL-S services [20] and other semantic data are added to the picture.

Conclusion and future work

We investigated different approaches for building universal mapping between the structure of the arbitrary XML documents and relational schema. Among different methods we selected the best one (Dynamic Level Numbering) and extend it to support XML documents with namespaces. Then we created our clean room implementation in Java of DNL method, and apply it for organizing a large collection of classified WSDL files and their semantic analogous (OWL-S files). Our implementation is not restricted to web services only and can be applied in many different scenarios where XML files are involved.

Finally we enumerate some of the improvements to our system that are needed to make it more robust and performant:

- use stream based XML parser as Stax;
- add full support of Xpath, now just basic subset of XPath is supported;
- use binary representation of DLN ids;
- create a more optimized database schema;
- more examples

References

1. K u c k e l b e r g, A., R. K r i e g e r. Efficient Structure Oriented Storage of XML Documents Using ORDBMS. – In: EEXTT and DIWeb 2002, LNCS 2590. S. Bressan et al. (Eds.). Springer Verlag, 2003, 131-143.
2. S h a n m u g a s u n d a r a m, J., E. J. S h e k i t a, J. K i e r n a n, R. K r i s h n a m u r t h y, S. V i g l a s, J. F. N a u g h t o n, I. T a t a r i n o v. A General Techniques for Querying XML Documents Using a Relational Database System. – In: SIGMOD Record , **30(3)**, 2001, 20-26.
3. F l o r e s c u, D., D. K o s s m a n n. Storing and Querying XML Data Using an RDBMS. – In: IEEE Data Engineering Bulletin, **22(3)**, 1999.
4. S c h m i d t, A., M. L. K e r s t e n, M. W i n d h o u w e r, F. W a a s. Efficient Relational Storage and Retrieval of XML Documents. – In: WebDB (Selected Papers) 2000, 137-150.
5. B a u e r, M. G., F. R a m s a k, R. B a y e r. Multidimensional Mapping and Indexing of XML. – In: Proc. of German Database Conf. BTW 2003, 305-323.
6. D i e t z, P. F. Maintaining Order in a Linked List. – In: Proc. of the 14th Annual ACM Symposium on Theory of Computing, California, 1982, 122-127.
7. S h i m u r a, T., M. Y o s h i k a w a, S. U e m u r a. Storage and Retrieval of XML Documents Using Object-Relational Databases. – In: Proc. of the 10th Intern. Conf. on Database and Expert Systems Applications (DEXA'99), LNCS 1677. Springer Verlag, 1999, 206-217.
8. L i, Q., B. M o o n. Indexing and Querying XML Data for Regular Path Expressions. – In: Proc. of the 27th VLDB Conf., Roma, Italy, 2001.
9. C h i e n, S., V. J. T s o t r a s, C. Z a n i o l o, D. Z h a n g. Storing and Querying Multiversion XML Documents Using Durable Node Numbers. – In: Proc. of the Intern. Conf. on WISE, Japan, 2001, 270-279.
10. L e e, Y. K., S. Y o o, K. Y o o n, P. B. B e r r a. Index Structures for Structured Documents. – In: Proc. of the 1st ACM Intern. Conf. on Digital Libraries, 1996, 91-99.
11. K h a, D. D., M. Y o s h i k a w a, S. U e m u r a. A Structural Numbering Scheme for XML Data. – In: EDBT 2002 Workshops, LNCS 2490. Chaudhri, A. B. et al. (Eds.). Springer Verlag, 2002, 91-108.
12. C o h e n, E., H. K a p l a n, T. M i l o. Labeling Dynamic XML Trees. – In: Proc. of PODS 2002.
13. O'N e i l, E., P. O'N e i l, S. P a l, I. C s e r i, G. S c h a l l e r, N. W e s t b u r y. ORDPATHs: Insert-Friendly XML Node Labels. ACM SIGMOD Industrial Track, 2004.
14. T a t a r i n o v, I., S. V i g l a s, K. S. B e y e r, J. S h a n m u g a s u n d a r a m, E. J. S h e k i t a, C. Z h a n g. Storing and Querying Ordered XML Using a Relational Database System. – In: Proc. of SIGMOD Conf., 2002, 204-215.

15. B o h m e, T., E. R a h m. Supporting Efficient Streaming and Insertion of XML Data in RDBMS. – In: Proc. 3rd Int. Workshop Data Integration over the Web (DIWeb), 2004.
16. Web Service Modeling Ontology
www.wsmo.org
17. **http://www.few.vu.nl/~andreas/projects/annotator/ws2003.html**
18. **http://www.salcentral.com/**
19. **http://www.xmethods.net/**
20. **http://www.few.vu.nl/~andreas/projects/annotator/owl-ds.html**