QUERYABLE SEPA MESSAGE COMPRESSION BY XML SCHEMA SUBTRACTION

Stefan Böttcher, Rita Hartel and Christian Messinger
Univeristy of Paderborn, Computer Science, Fürstenallee 11, 33102 Paderborn, Germany

Keywords: SEPA-XML Message Compression, SEPA Data, Exchange, Efficient Query Processing on Compressed SEPA Data.

Abstract: In order to standardize the electronic payments within and between the member states of the European Union, SEPA (Single Euro Payments Area) – an XML based standard format – was introduced. As the financial institutes have to store and process huge amounts of SEPA data each day, the verbose structure of XML leads to a bottleneck. In this paper, we propose a compressed format for SEPA data that removes that data from a SEPA document that is already defined by the given SEPA schema. The compressed format allows all operations that have to be performed on SEPA data to be executed on the compressed data directly, i.e., without prior decompression. Even more, the queries being used in our evaluation can be processed on compressed SEPA data with a speed that is comparable to ADSL2+, the fastest ADSL standard. In addition, our tests show that the compressed format reduces the data size to 11% of the original SEPA messages on average, i.e., it compresses SEPA data 3 times stronger than other compressors like gzip, bzip2 or XMill – although these compressors do not allow the direct query processing of the compressed data.

1 INTRODUCTION

1.1 Motivation

In order to simplify and to standardize the inner-European financial infrastructure, the European Payment Council (EPC) applied the XML-based standard SEPA (Single Euro Payments Area) that defines a format for financial transactions amongst the member states, i.e., amongst all members of the European Union plus Liechtenstein, Iceland, Norway, Monaco and Switzerland. For example, since January 2008 it is possible to execute money transfers in SEPA format and by the end of 2012, all SEPA members will have to replace all their national payment systems by SEPA payment systems.

As the SEPA format specifies customer-to-bank money transactions ("pain" messages) as well as inter-bank money transactions ("pacs" messages) each bank has to process and store huge amounts of XML data.

1.2 Contributions

In this paper we present an approach to XML compression – called XML Schema subtraction (XSDS) – that allows compressing the XML structure of SEPA messages into a data format that is 9 times smaller than the original message size. Furthermore, XSDS allows to process the compressed messages in a similar way as the original messages – e.g. by evaluating XPath queries – without prior decompression.

Thus, using XSDS-compressed SEPA messages instead of original SEPA messages as internal format within a bank institute allows on the one hand to save storage costs while archiving the data and on the other hand to reduce the amount of data to be processed.

1.3 Paper Organization

The remainder of this paper is organized as follows. Section 2 describes the basic concept XSDS, i.e. how schema information can be removed from an XML document. Section 3 gives an overview of how the compressed data can be processed directly, i.e., without prior decompression. Section 4 evaluates the

Böttcher S., Hartel R. and Messinger C. (2010).
QUERYABLE SEPA MESSAGE COMPRESSION BY XML SCHEMA SUBTRACTION.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Software Agents and Internet Computing*, pages 23-29
DOI: 10.5220/0002889000230029
Copyright © SciTePress

compression ratio of XSDS and query processing on XSDS compressed data based on SEPA data. Section 5 compares XSDS to related work. Finally, Section 6 summarizes our contributions.

2 THE CONCEPT

2.1 The Basic Idea

SEPA is a standard that defines the format of electronic payment within the member states of the EU. Each electronic payment is processed and stored in form of an XML document, the format of which is defined by a set of XML schemata (XSD) by SEPA.

Some parts of each SEPA file are strictly determined by the SEPA standard, e.g., that each payment message starts with the tag <Document> or that the first two child nodes of the element <GrpHdr> (group header) are the elements <MsgId> (message ID) and <CreDtTm> (date and time of message creation). Other parts are variable and vary from document to document (e.g. whether the Debtor (<Dbtr>) has a postal address (<PstAdr>) or not).

The main compression principle of XML schema subtraction (XSDS) is the following. XSDS removes all information that is strictly defined by the XML schema information from a given XML document, and, in the compressed format, XSDS encodes only those parts of the XML document that can vary according to the XML schema. The compression principle of XSDS is similar to the compression principles of XCQ (Ng et al., 2006) and DTD subtraction (Böttcher, Steinmetz, and Klein, 2007) which are able to remove information provided by a DTD from a given XML document. However, in contrast to these approaches, XSDS removes information given by an arbitrary XML schema, which is significantly more complex than just considering DTDs. The current paper reports about XSDS, but focuses on the advantages of applying XSDS to SEPA as an application standard which is significant for financial transactions in the EU member states.

2.2 This Paper's Example

As the whole SEPA standard is too huge to be discussed within this paper, we only have a detailed look on a small excerpt. Each payment document contains (amongst others) a Debtor. The information on the Debtor is stored as an element with label <Dbtr> that contains a name (label <Nm>) and zero or one ID (label <Id>) followed by zero or one postal address (label <PstAdr>). The ID contains either a privat ID (label <PrvtId>) or an organization ID (label <OrgId>). The postal address consists of a city (label <City>) and zero to two address lines (label <AdrLine>). Figure 1 shows a graphical visualization of the element <Dbtr> and its definition.

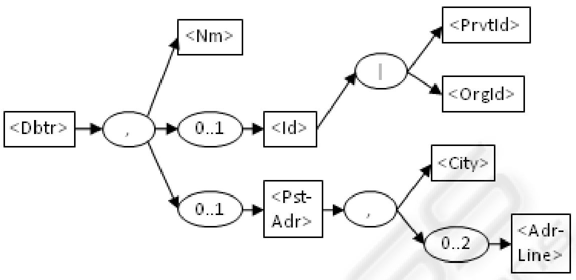


Figure 1: Excerpt of the SEPA schema.

2.3 Removing Schema Information from the Document Structure in XSDS

Within the structure of an XML document, i.e., within the element tags, there exist only three different concepts that allow for variant parts within an XML document defined by a given schema: First, the XSD requires the choice of one out of different given alternatives. Second the XSD element 'all' requires the occurrence of all elements declared by children of the 'all' element, but they can occur in any order. Third, when the XSD requires a repetition of elements, this usually allows for a varying number of elements (including all its descendant elements).

The compression of these variant parts within an XML document works as follows. Each compression step assumes that we consider one current position in the XML document at a time for which the XSD allows variant parts. For each current position in the XML document for which the XSD allows a choice, we only store the alternative chosen at the current position. (This requires log(n) bits, if there are n possible alternatives). For each XSD element 'all', we only encode the order of the elements required by the children of the 'all' element in the XSD. Finally, for each repetition of elements starting at a given position within an XML document, we only store the number of occurrences of this element found at the current position of the XML document. (If the number of children per node is e.g. limited by 2^32 (MAXINT), this requires 1-5 bytes per repetition node, depending on the concrete number of repetitions).

The compression of the non-variant parts of an XML document, i.e. of the nodes that are fixed by the XSD for the current position in the XML document, is even much simpler. We can omit these nodes from our compressed format, as these nodes can be reconstructed for the given position from the XSD.

When applying the compression by removing schema information from the SEPA excerpt shown in Figure 1, we do the following for the variant parts of a given XML document fragment matching this schema excerpt. We store a single bit for the repetition nodes with label '0..1' stating whether or not there is an <Id> element and whether or not there is an <PstAdr> element found in the current XML document position. We store a single bit for the choice node with label '|' stating whether there is a <PrvtId> or a <OrgId> element found in the current XML document position. Finally, we store two bits for the repetition node with label '0..2' stating whether there are 0, 1, or 2 elements <AdrLine>.

The remaining parts of an XML fragment for this SEPA excerpt are fixed by the SEPA excerpt. This includes all element names found in the XML fragment. For example, not only the element name <Dbtr> of the fragment is root is fixed, but also the element name <Nm> of the first child of the <Dbtr> element is fixed. Furthermore, it is not necessary to include the element names for optional parts like <Id> or <PstAdr> - when the option has been chosen, the element name is fixed by the XSD. Similarly, for repetitions, e.g. occurrences of the <AdrLine> element, it is sufficient to store the number of repetitions in the compressed data format. The element name for each repeated element is fixed. Finally, the <PrvtId> element must occur whenever the first alternative of the choice is taken.

Therefore, we need (at most) 5 bits to store the structure of each possible XML fragment matching the structure of a <Dbtr>-element in the given SEPA excerpt. Requiring only 5 bits is optimal, as there exist 24 different 'shapes' of the <Dbtr>-element and its descendants.

2.4 Compressing the Textual Data

Beneath the structure, a SEPA document contains textual data. Whereas large parts of the structure are defined by the schema, less information is given on the textual data. Nevertheless, compression of textual data and query evaluation on compressed data can be improved by grouping together textual data that is included by the same parent elements.

For these purposes, for each parent element of textual data, a single container is provided that stores the textual data in document order. Storing the textual data in different containers provides two advantages:

- When processing the SEPA documents, different queries have to be evaluated, as e.g. whether the payment creditor is on an embargo list. In many cases, this can be answered by simply searching in a few containers.
- As each container contains data of the same domain (e.g., names, zip codes …) compressing each container separately from the other containers yields a stronger compression ratio than compressing all the textual data of one document together.

XSDS mainly differs between three different types of textual data: String data, Integer data and data enumerations that only allow a value of a given enumeration of possible values (e.g., the address type of a postal address of an invoicee can be one value of the following list: ADDR, PBOX, HOME, BIZZ, MLTO, DLVY).

In our implementation, each container that contains String data is compressed via the generic text compressor gzip. Each entry of an Integer container is stored via a variable-length Integer encoding that stores 1-5 bytes per Integer value depending on the concrete value. Finally, each value stored within an enumeration container is stored analogously to a choice within the structure: If an enumeration container allows n different values, each value is represented by an encoding with size log(n) bits, that defines the position of the current value within the alternative.

3 QUERY PROCESSING

In contrast to other compressors like XMill (Liefke and Suciu, 2000), gzip or bzip2 that are mainly used for archiving, XSDS is able to evaluate queries on the compressed data directly, i.e., without prior decompression.

This makes XSDS not only useful for archiving data, but also for compressing data that is still processed or exchanged among partners. For example, for a bank institute, this means that the bank institute compresses SEPA data that it receives, if it is not already compressed. Then, the bank institute can process the compressed SEPA data and archive it without any need of decompression or recompression between multiple processing steps. Only if the bank institute sends the data to a customer or another institute that requires uncompressed SEPA data as input, a decompression into the uncompressed SEPA format might be needed.

For query evaluation on compressed SEPA data, we use the looking forward approach (Olteanu, et. al., 2002) followed by a query rewriting system that reduces queries to using only the axes child, descendant-or-self, following-sibling and a self axis using filters (citation omitted to avoid self reference). Additionally, we have implemented a generic query processing engine that further reduces XPath queries to queries on the basic axes first-child, next-sibling, and parent, and the operations getXMLNodeType and getLabel on the data compressed by our XSDS compression approach.

In order to determine first-child, next-sibling or label of a current context node, we simultaneously parse through the XML schema and the compressed document. Similar to keeping track of the current context node which describes the actual parsing position in the XML document, simultaneously parsing keeps track of a current XML schema node which describes the actual parsing position in the XML schema. Whenever the XML schema allows for variant parts, as there is a repetition, a choice or an ‘all’ element in the schema, the concrete choice selected for this variant part is looked up in the compressed data.

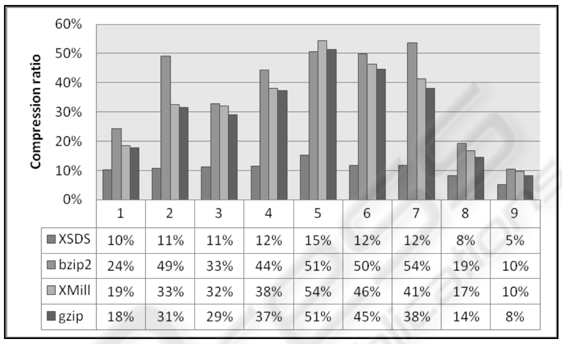
Concerning the textual data, only those containers have to be decompressed that contain data that is addressed by the query: either data that is needed for evaluating a predicate filter or data that is part of the output result. Whenever the compressor of a data container allows partial decompression of the compressed data and value comparisons directly in the compressed data, no needless decompression of textual data is performed at all. Only those text values that are required for query processing are decompressed and read.

4 EVALUATIONS

4.1 Compression Ratio

In order to evaluate our approach, we have collected 9 SEPA example files provided by different bank institutes from the internet. We have compared our approach to 3 different compression approaches: First, Gzip – a generic text compressor based on LZ77 and Huffman, second XMill (Liefke and Suciu, 2000) – a non-queryable XML compressor, and third, bzip2 – a generic text compressor based on Burrows-Wheeler Transform (Burrows and Wheeler, 1994), Move-to-Front and Huffman.

The results of our evaluation are shown in Figure 2. Although all other tested compressors do not allow query evaluation on the compressed data, i.e., they require a prior decompression when processing the data, our approach additionally outperforms them in terms of the reached compression ratio, i.e. the size of compressed document divided by the size of original document.



	1	2	3	4	5	6	7	8	9
■ XSDS	10%	11%	11%	12%	15%	12%	12%	8%	5%
■ bzip2	24%	49%	33%	44%	51%	50%	54%	19%	10%
■ XMill	19%	33%	32%	38%	54%	46%	41%	17%	10%
■ gzip	18%	31%	29%	37%	51%	45%	38%	14%	8%

Figure 2: Compression ratio reached for SEPA documents.

While bzip2 reaches compression ratios from 8% to 51% (37% on average), gzip reaches compression ratios from 10% to 49% (30% on average) and XMill reaches compression ratios of 10% to 54% (32%) on average, XSDS reaches compression ratios of 5% to 15% (11% on average). In other words, on average XSDS compresses 3 times stronger than all other evaluated compressors. To the best of our knowledge, XSDS is the compressor that reaches the strongest compression on SEPA documents.

4.2 Query Performance

In order to test the query performance, we have generated a set of example SEPA documents with the same structure, but increasing size (while the smallest document, D12, has a size of 17 kB and contains 2 SEPA messages, the largest document, D1, has a size of 193 MB and contains 25,000 SEPA messages).

We have evaluated the following set of queries that ask for debtor names, currency of the payment, a complete SEPA message, or the amounts of the payments on the documents:

Q1=/sepade/Msg/Document/pain.001.001.02
/PmtInf/Dbtr/Nm
Q2= //Dbtr/Nm
Q3= //InstdAmt[@Ccy]
Q4= /sepade/Msg
Q5=//Amt

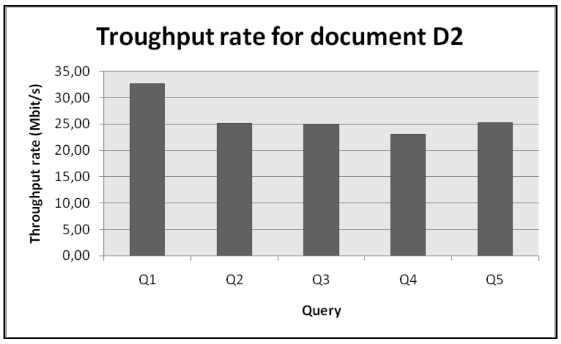


Figure 3: Throughput reached for document D2.

Figure 3 shows the results of our evaluation on document D2 (77 MB, 10,000 SEPA messages). For this document, our query evaluation on the compressed document reaches an average throughput rate which is equivalent to 26 Megabits/s regarding the uncompressed original SEPA document. In other words, our query evaluation is faster than the ADSL2+ - currently the fastest ADSL standard – that reaches download rates of 24 Mbit/s.

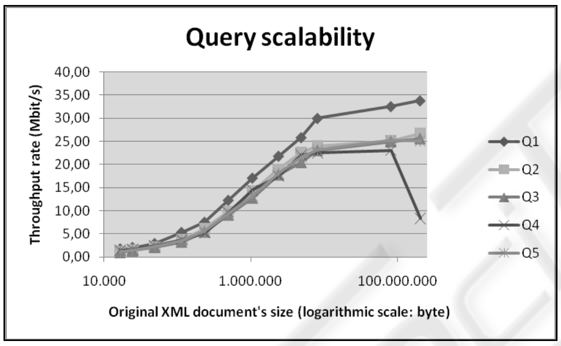


Figure 4: Query scalability.

In order to test the scalability of the query evaluation on the XSDS compressed data, we have evaluated the queries Q1 to Q5 on the documents D1 to D12. As shown in Figure 4, query evaluation on XSDS compressed data scales excellently, as the throughput rate even increases with increasing the original file size up to about 100 MB. One reason for the increase of the throughput rate up to an original file size of 100 MB is that the reached compression ratio is stronger, the bigger the files are, and thus the data volume to be processed during query evaluation decreases in relation to the size of the original file. For an original file size of more than 100 MB, the average query evaluation throughput decreases at least for query Q4 (that requires a total decompression of all text zipped text containers).The reason for the final decrease of the query evaluation throughput lies within the size of the text containers. The bigger the zipped text containers get, the less efficient they can be accessed. Because of that, we propose to split the text containers into several text containers as soon as a certain threshold (e.g. 10,000 SEPA messages) is exceeded. As each SEPA message represents a single payment transaction, no queries have to be evaluated that span several SEPA messages. Therefore, splitting a text container when its size exceeds a given threshold will not lead to any restriction.

5 RELATED WORK

There exist several approaches to XML structure compression, which can be mainly divided into three categories: encoding-based compressors, grammar-based compressors, and schema-based compressors.

Many compressors do not generate compressed data that supports evaluating queries, i.e., any query processing on the compressed data needs prior decompression.

The encoding-based compressors allow for a faster compression speed than the other ones, as only local data has to be considered in the compression as opposed to considering different sub-trees as in grammar-based compressors.

The XMill algorithm (Liefke and Suciu, 2000) is an example of the first category. The structure is compressed, by assigning each tag name a unique and short ID. Each end-tag is encoded by the symbol '/'. This approach does not allow querying the compressed data.

XGrind (Tolani and Hartisa, 2002), XPRESS (Min, Park, and Chung, 2003) and XQueC (Arion et al., n.d.) are extensions of the XMill approach. Each of these approaches compresses the tag information using dictionaries and Huffman-encoding (Huffman, 1952) and replaces the end-tags by either a '/'symbol or by parentheses. All three approaches allow querying the compressed data. However, as all of them result in a weaker compression than XMill, XSDS compresses stronger than all of them.

The encoding-based compression approaches (Bayardo et al., 2004), (Cheney, 2001), and (Girardot and Sunderesan, 2000) use tokenization. (Cheney, 2001) replaces each attribute and element name by a token, where each token is defined when it is being used for the first time. (Bayardo et al., 2004) and (Girardot and Sunderesan, 2000) use tokenization as well, but they enrich the data by additional information that allows for a fast navigation (e.g., number of children, pointer to next-sibling, existence of content and attributes). All three approaches use a reserved byte for encoding the end-tag of an element. They all allow querying the compressed data.

The encoding-based compression approach in (Zhang, Kacholia, and Özsu, 2004) defines a succinct representation of XML that stores the start-tags in form of tokens and the end-tag in form of a special token (e.g. ')'). They enrich their compressed XML representation by some additional index data that allows a more efficient query evaluation. This approach allows querying of compressed data.

XQzip (Cheng and Ng, 2004) and the approaches presented in (Adiego, Navarro, and de la Fuente) and (Buneman, Grohe, and Koch, 2003) belong to grammar-based compression. They compress the data structure of an XML document by combining identical sub-trees. Afterwards, the data nodes are attached to the leaf nodes, i.e., one leaf node may point to several data nodes. The data is compressed by an arbitrary compression approach. These approaches allow querying compressed data.

An extension of (Buneman, Grohe, and Koch, 2003) and (Cheng and Ng, 2004) is the BPLEX algo-rithm (Busatti, Lohrey, and Maneth, 2005). This approach does not only combine identical sub-trees, but recognizes similar patterns within the XML, and therefore allows a higher degree of compression. It allows querying of compressed data.

Schema-based compression comprises such approaches as XCQ (Ng et al., 2006), XAUST (Subramanian, and Shankar, 2005), Xenia (Werner et al., 2006) and DTD subtraction (Böttcher, Steinmetz, and Klein, 2007). They subtract the given schema information from the structural information. Instead of a complete XML structure stream or tree, they only generate and output information not al-ready contained in the schema information (e.g., the chosen alternative for a choice-operator or the num-ber of repetitions for a *-operator within the DTD). These approaches are queryable and applicable to XML streams, but they can only be used if schema information is available.

XSDS follows the same basic idea to delete information which is redundant because of a given schema. In contrast to XCQ, XAUST and DTD subtraction that can only remove schema information given by a DTD, XSDS works on XML schema which is significantly more complex than DTDs. Furthermore, XSDS uses a counting schema for repetitions that compresses stronger than e.g. the ones used in XCQ or Xenia.

The approach in (Ferragina et al., 2006) does not belong to any of the three categories. It is based on Burrows-Wheeler Transform (Burrows and Wheeler, 1994), i.e., the XML data is rearranged in such a way that compression techniques such as gzip achieve higher compression ratios. This approach allows querying the compressed data only if it is enriched with additional index information.

In comparison to all other approaches, XSDS is the only approach that combines the following advantageous properties: XSDS removes XML data nodes that are fixed by the given XML schema, it encodes choices, repetitions, and 'all'-groups in an efficient manner, and it allows for efficient query processing on the compressed XML data.

To the best of our knowledge, no other XML compression technique combines such a compression performance for SEPA data with such query processing speed on compressed data.

6 CONCLUSIONS

We have presented XSDS (XML schema subtraction) – an XML compressor that performs especially well for electronic payment data in SEPA format.

XSDS removes all data that can be inferred from the given schema information of the XML document. Thereby, XSDS provides two major advantages: First, XSDS generates a strongly compressed document representation which may save costs and energy by saving bandwidth for data transfer and by saving main memory required to process data and by saving secondary storage needed to archive compressed XML data. Second, XSDS supports fast query evaluation on the compressed document without prior decompression.

Our experiments have shown that XSDS compresses SEPA messages down to a size of 11% of the original SEPA document size on average, which outperforms the other compressors, i.e. gzip, XMill and bzip2, by a factor of 3. Furthermore, query evaluation directly on the compressed SEPA data is not only possible, but in our experiments, query processing reaches throughput rates that are higher than those of ADSL2+. Therefore, we consider the XSDS compression technique to be highly beneficial in all SEPA applications for which the data volume is a bottleneck.

REFERENCES

J. Adiego, G. Navarro, P. de la Fuente: Lempel-Ziv Compression of Structured Text. Data Compression Conference 2004

A. Arion, A. Bonifati, I. Manolescu, A. Pugliese. XQueC: A Query-Conscious Compressed XML Database, to appear in ACM Transactions on Internet Technology.

R. J. Bayardo, D. Gruhl, V. Josifovski, and J. Myllymaki., 2004. An evaluation of binary xml encoding optimizations for fast stream based XML processing. In Proc. of the 13th international conference on World Wide Web.

S. Böttcher, R. Steinmetz, N. Klein, 2007. XML Index Compression by DTD Subtraction. International Conference on Enterprise Information Systems (ICEIS).

P. Buneman, M. Grohe, Ch. Koch, 2003. Path Queries on Compressed XML. VLDB.

M. Burrows and D. Wheeler, 1994. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation.

G. Busatto, M. Lohrey, and S. Maneth, 2005. Efficient Mem¬ory Representation of XML Dokuments, DBPL.

K. S. Candan, W.-P. Hsiung, S. Chen, J. Tatemura, D. Agrawal, 2006. AFilter: Adaptable XML Filtering with Prefix-Caching and Suffix-Clustering. VLDB.

J. Cheney, 2001. Compressing XML with multiplexed hierarchical models. In Proceedings of the 2001 IEEE Data Compression Conference (DCC 2001).

J. Cheng, W. Ng: XQzip, 2004. Querying Compressed XML Using Structural Indexing. EDBT.

P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan, 2006. Compressing and Searching XML Data Via Two Zips. In Proceedings of the Fifteenth Interna¬tional World Wide Web Conference.

M. Girardot and N. Sundaresan. Millau, 2000. An Encoding Format for Efficient Representation and Exchange of XML over the Web. Proceedings of the 9th International WWW Conference.

D.A. Huffman, 1952. A method for the construction of minimum-redundancy codes. In: Proc. of the I.R.E.

J. Ziv and A. Lempel: A Universal Algorithm for Sequential Data Compression, 1977. In IEEE Transactions on In¬formation Theory, No. 3, Volume 23, 337-343

H. Liefke and D. Suciu, 2000. XMill: An Efficient Compres¬sor for XML Data, Proc. of ACM SIGMOD.

J. K. Min, M. J. Park, C. W. Chung, 2003. XPRESS: A Queriable Compression for XML Data. In Proceedings of SIGMOD.

W. Ng, W. Y. Lam, P. T. Wood, M. Levene, 2006: XCQ: A queriable XML compression system. Knowledge and Information Systems.

D. Olteanu, H. Meuss, T. Furche, F. Bry, 2002: XPath: Looking Forward. EDBT Workshops.

H. Subramanian, P. Shankar: Compressing XML Documents Using Recursive Finite State Automata. CIAA 2005

P. M. Tolani and J. R. Hartisa, 2002. XGRIND: A query-friendly XML compressor. In Proc. ICDE.

Ch. Werner, C. Buschmann, Y. Brandt, S. Fischer: Compressing SOAP Messages by using Pushdown Automata. ICWS 2006

N. Zhang, V. Kacholia, M. T. Özsu, 2004. A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML. ICDE