Building a Semantic Web of Things: issues and perspectives in information compression

Floriano Scioscia
Politecnico di Bari
via Re David 200, I-70125 Bari, ITALY
f.scioscia@poliba.it

Michele Ruta
Politecnico di Bari
via Re David 200, I-70125 Bari, ITALY
m.ruta@poliba.it

Abstract

The paper surveys some relevant issues related to data management in pervasive environments. Particularly, it focuses on the compression of semantic annotations for building so called Semantic Web of Things. An approach is proposed to achieve good compression ratios, to maintain compression effectiveness and finally to query compressed data without requiring expensive decompression phases. Experimental results prove the goodness of the proposed framework.

1 Introduction

The *Semantic Web of Things* (SWoT) is an emerging vision in Information and Communication Technology, joining together some of the most important ICT paradigms of the decade: the Semantic Web and the Internet of Things.

The *Semantic Web* initiative [2] aims at allowing available information in the World Wide Web to be seamlessly shared, reused and combined by software agents. Each available resource in the semantic-enabled Web should be annotated, using RDF[1], w.r.t. an OWL ontology [2]. Both RDF and OWL are defined through an XML Schema[3].

The *Internet of Things* vision [7] promotes on a global scale the ubiquitous computing paradigm. In ubiquitous and pervasive contexts, intelligence is embedded into objects and physical locations by means of a relatively large number of heterogeneous micro-devices, each conveying a small amount of useful information. Such information should be automatically extracted and processed by mobile devices, in order to better support the current activity of a user and satisfy her needs.

This paper focuses on the compression of semantic annotations, a relevant technical aspect for SWoT architectures. XML formats adopted in the Semantic Web are too verbose to allow efficient data storage and management in mobile environments. Compression techniques become essential in order to enable storage and transmission of semantically annotated information on tiny mobile devices such as *Radio Frequency IDentification* (RFID) tags or wireless sensors. Moreover, the benefits of compression will apply to the whole ubiquitous computing environment, as decreasing data size means shorter communication delays, efficient usage of bandwidth and reduced battery drain.

Manifold aspects should be analyzed when evaluating approaches to compression for the SWoT: *compression ratio*; *compression efficiency* (particularly, time and main memory usage are critical, the former due to the highly dynamic nature of pervasive contexts, the latter because current mobile computing devices have low memory amounts and implement less sophisticated memory management techniques than PCs); *efficiency and effectiveness of queries on compressed data* (recent research has been increasingly focused on compression schemes for XML documents allowing to directly query encoded annotations [10] which would eliminate the need for decompression before query evaluation).

The remainder of the paper is organized as follows. Section 2 describes the architectural framework for enabling a Semantic Web of Things, introducing the role of compression. In Section 3 several solutions are discussed for the compression of XML-based document instances, and a novel approach is introduced, using DIG [1] as reference language. Section 4 reports a performance comparison of the discussed algorithms. Conclusions are presented in Section 5.

[1]RDF Primer, W3C Recommendation, February 10th 2004, http://www.w3.org/TR/rdf-primer/

[2]OWL Web Ontology Language, W3C Recommendation, February 10th 2004, http://www.w3.org/TR/ owlfeatures/

[3]XML Schema Primer, W3C Recommendation, October 28th 2004, http://www.w3.org/TR/xmlschema-0/

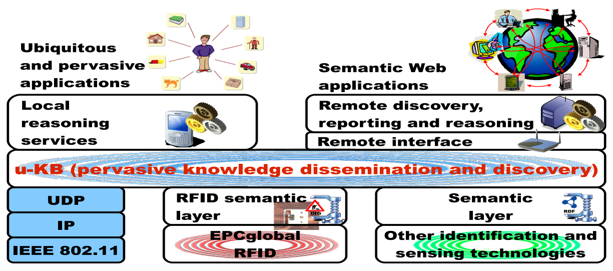
Ubiquitous and pervasive applications
Semantic Web applications
Local reasoning services
Remote discovery, reporting and reasoning
Remote interface
u-KB (pervasive knowledge dissemination and discovery)
UDP
IP
IEEE 802.11
RFID semantic layer
EPCglobal RFID
Semantic layer
Other identification and sensing technologies

Figure 1. Proposed SWoT architecture

2 Architecture for a Semantic Web of Things

The goal of the Semantic Web of Things is to embed semantically rich and easily accessible information into the physical world. To this aim, Knowledge Representation tools and technologies must be adapted to functional and non-functional requirements of mobile computing applications.

In [12] we proposed a possible architecture for the SWoT (illustrated in Figure 1), based on a Knowledge Base model, named *ubiquitous Knowledge Base* (u-KB). The u-KB layer provides common access to information embedded into semantic-enhanced micro-devices (such as sensors or RFID tags) populating a pervasive environment. It comprises the specification of components and operations of a u-KB, as well as a distributed application-layer protocol for fully decentralized dissemination and discovery of knowledge embedded within micro-devices. As reference formalism for resource annotation and matchmaking we exploit ontology languages originally conceived for the Semantic Web effort, particularly DIG [1], which is a more compact equivalent of OWL-DL. As figure shows, information processing and reasoning tasks can be performed either by local hosts (so enabling semantic-enhanced ubiquitous applications) or by a remote entity so integrating the local environment with the WWW. Reuse of Semantic Web standards is the key to simplify the integration of u-KBs in larger information infrastructures.

In a u-KB, knowledge is scattered within a smart environment, as KB individuals are physically tied to micro devices deployed in the field. For example, in RFID-based scenarios, each individual is represented by a semantically annotated object/product description, stored within the RFID transponder the object is clung to. Each annotation refers to an ontology providing the conceptual model for the specific domain. Figure 2 reports a very short example of DIG document, describing a delivery of refrigerated cow milk w.r.t. a given ontology for the food products domain. Current identification and sensing technologies require a semantic adaptation in order to define data formats and storage schemes to host such semantically annotated fragments on individual tiny devices. An algorithm for compressing semantic-based annotations is a basic part of this adaptation effort.

```
<tells xmlns="http://dl.kr.org/dig/2003/02/lang">
  <defindividual name="milk_delivery_M2"/>
  <instanceof>
    <individual name="milk_delivery_M2"/>
    <and>
      <catom name="cow_milk"/>
      <all>
        <ratom name="processed_with"/>
        <catom name="refrigeration"/>
      </all>
    </and>
  </instanceof>
</tells>
```

Figure 2. DIG document example

3 Compression approaches and tools

In what follows we focus on compression approaches and tools. First of all, a general background is provided, discussing the most widespread general-purpose and XML-specific algorithms and their relevant properties. Afterwards the proposed solutions are presented.

3.1 Background

Lossless compression is based on substitution of symbols (or groups of symbols) with code words, which is performed according to a statistical model for the input source. Based on the reference model, algorithms can be categorized into three families: static, semi-adaptive, and adaptive [5]. *Static* compression uses either fixed statistics or no statistics. *Semi-adaptive* compression works in two steps: the input data is scanned once for statistics collection and again for encoding. In *adaptive* compression, instead, statistics are gathered and dynamically updated along with compression.

gzip[4] is a very popular universal compression tool. It is based on a variant of the *LZ77* Ziv-Lempel algorithm [15]. Basically, it searches for character sequences recurring at least once within the file to be compressed. Then it substitutes further occurrences with the pair: *distance* (in byte) from the first instance, *length* of the sequence itself.

Huffman encoding [6] and ***arithmetic encoding*** [14] are semi-adaptive compression techniques. The former uses a variable-length code table (the *huffman tree*), determined by evaluating the frequency of each symbol (*i.e.*, the source model). The huffman tree is built so that the most common symbols are encoded using bit strings shorter than less common ones. In arithmetic encoding, instead, the whole

[4]GZIP compression utility: http://www.gzip.org/

Algorithm 1 $arithmetic\_encoding\,(m, subint)$

```
Require: $m$ message to be encoded, $subint$ associative array of sub–intervals assigned to symbols
Ensure: $n \in [0,1)$ real number that encodes $m$
1: $(min, max) := (0, 1)$
2: while $has\_next\_symbol(m)$ do
3:     $s := get\_next\_symbol(m)$
4:     $(begin, end) := subint[s]$
5:     $min := min + (max - min) * begin$
6:     $max := min + (max - min) * end$
7: end while
8: $n := min$
```

message is represented by a real value between 0 and 1. Disjoint sub-intervals of the $[0, 1)$ are assigned to all possible symbols: the length of each sub-interval depends on its frequency. Then the encoding of a message is performed as in Algorithm 1. At the beginning, the message corresponds to the whole $[0, 1)$. Then each symbol of it progressively reduces the size of the interval proportionally to its sub-interval. In this class of algorithms and tools, the ***PAQ*** family [9] currently has the best compression rates, but its processing and memory requirements are far beyond the capabilities of mobile computing devices.

Higher compression rates can be achieved by algorithms specifically designed for XML encoding. An important feature of XML compression is *homomorphism* [13]. It preserves the structure of the original XML data. On the contrary, with non-homomorphic algorithms the document structure is not recognizable after compression. Homomorphism may allow to evaluate queries directly on compressed formats, by detecting document pieces satisfying query conditions without a preliminary decompression.

XMill [8] is an efficient schema-aware XML compressor. It is based upon splitting XML content into different *containers*, which are separately compressed and sequentially stored in the output file (hence XMill is non-homomorphic). XMill performances are better than generic compressors for medium and large XML documents, whereas for small files (up to 20 kB approximately) they are penalized by the small size of containers. Instead, ***DPDT*** [4] adopts a syntactic approach to XML compression. A DTD or XML Schema is considered as a *Dictionary Grammar*, which is a variant of a Context Free Grammar. The document is encoded using *Partial Prediction Matching* (PPM), an adaptive encoding technique. Obtained compression rates are higher than XMill, though PPM-based compressors are generally slower.

XPRESS [10] exploits *reverse arithmetic encoding*, a semi-adaptive algorithm. In the first step, statistics are gathered and each XML tag is mapped to a numeric interval proportional to its frequency. In the second step, arithmetic encoding is applied: the message to be encoded is the sequence of tag labels, starting with a given tag and going toward its ancestors up to the root tag (hence the adjective "reverse"). Furthermore, a *type inference* component detects data types of XML attribute values and a specialized encoding method is applied for each of them (numbers, dates, etc.). Experimental results showed XPRESS achieves high query performance for XML data and it allows updates to be performed directly on compressed data. Compression rates, though, are lower than other methods.

In [3] we proposed a simple compression algorithm and tool for DIG document instances, named ***DIGCompressor***. Experimental tests in a Bluetooth-based testbed and RFID simulations that evidenced its suitability to pervasive contexts. DIGCompressor exploits the peculiarities of the DIG XML Schema, which comprises at most 40 different tags; no text is allowed inside any tag and only specific tag attributes are permitted. Pure *data* (XML elements and attributes) and *data structures* (attribute values) are encoded in different ways. Three fundamental compression stages are performed. **(1)** ***Data structures packing***. It encodes the DIG document structure. A unique 8-bit code is statically associated to each element or attribute in the DIG XML Schema. Compression is homomorphic. **(2)** ***Attribute values packing***. Most recurrent attribute value strings are identified and encoded with 16-bit sequences. A header of the compressed file is thus built, containing the decoding table. Therefore this compression step is semi-adaptive. Since a simple code substitution is applied, homomorphism is obeyed. As the use of the header could lower compression performance for short files, a heuristic rule was adopted: only attributes above both a length threshold and a frequency threshold are encoded. **(3)** ***Zlib packing***. Finally *zlib* library[5] is exploited to apply a final compression level. It is based on the *Ziv-Lempel* compression algorithm, which is adaptive and non-homomorphic, as said above.

3.2 COX

Here we introduce a novel approach for the compression of XML-based semantic annotations. Its main goal is to support queries directly on compressed data. To this aim, Reverse Arithmetic Encoding (RAE), a semi-adaptive and homomorphic technique (also used in XPRESS [10]) was adapted. The approach was implemented into a pair of tools called ***COX*** (Compressor for Ontological XML-based languages) and ***DECOX*** (DECompressor), respectively. They can be used for documents in RDF, DIG and OWL.

Similarly to DIGCompressor, two different solutions are adopted to encode data structures and data. For data structures (XML tags and attributes), RAE is used. For the document data section, the most recurrent words are encoded like in DIGCompressor. Both techniques require a two-step process. In the first step the XML document is parsed and

[5]ZLIB library, available at http://www.zlib.net

statistics are gathered. In the second step compression is performed and output is written.

First step. DIG and OWL languages only contain tags, attributes and attribute values (excluding other syntax elements such as comments and processing instructions). COX deals with both tag and attribute names in the same way, only distinguishing the latter by means of a "`@`" prefix added to the name. Therefore, from now on with the word "tag" we will refer either to tags or attributes. After parsing, an adjusted frequency of each tag name is computed as ratio between the number of occurrences of the tag and the total document tags. The interval $[d, D) = [1.0 + 2^{-7}, 2.0 - 2^{-15})$ is split in disjoint sub-intervals associated to each single tag. In [10] the length of each sub-interval is proportional to the adjusted frequency of the tag, but our experimental evaluations proved this approach can lead to errors in decompression for tags with a very low frequency (in the order of 10^{-4}). Hence, we designed a new sub-interval calculation function, to assign slightly longer sub-intervals to very rare tags while preserving proportionality with respect to frequency:

$a_i = kf_i + \frac{1-k}{n}$

where a_i is the length of the i-th sub-interval, f_i is the adjusted frequency of the i-th tag, n is the number of different tag names occurring in the document. The weight k is defined as:

$k = 1 - \sqrt{\sigma}$

where σ is the standard deviation of the absolute frequencies of all the tags in the document. The formula is a linear combination of two components: the first one is proportional to tag frequency, the second one is fixed for all tags. When the frequency distribution of tags in a document is regular, $\sigma \to 0$ and $k \to 1$, so the proportional term dominates and COX behaves very similarly to XPRESS. Conversely, for irregular frequency distributions (the most common case) σ increases and k decreases, so that the second term of the formula increases and even very rare tags receive a not-too-small sub-interval. The minimum value of the sub-interval assigned to each tag is computed with the formula:

$min_{i+1} = min_i + a_i * (D - d), i = 1, \ldots, n - 1$

where n is total number of tags and $min_1 = d$. All values that represent opening tags fall in the interval $[d, D)$. The interval $[1.0, d)$ is reserved to encode closing tags. Notice that every possible value is strictly between 1.0 and 2.0. Using 32-bit floating point representation, the first byte will always be $\texttt{01111111}_2$, so it can be truncated without loss of information [10]. After the first step a header is written at the beginning of the output file. It contains a sequence of records composed by: 1 byte for the length of the tag name; the tag name; 3 bytes (after truncation) for the encoding of the minimum value of the sub-interval of the tag. Attribute values are simply interpreted as ASCII strings. The statistic collection is performed concurrently w.r.t. the analysis of document structure. Exploited technique and reference criteria are as in the second step of DIGCompressor. Another header is written which merely is a sequence of ASCII strings of attribute values, separated by the $\texttt{ff}_h$ character. The corresponding codes are single-byte values from $\texttt{00}_h$ to $\texttt{fd}_h$ and they are assigned to strings in progressive order, hence they can be omitted in the header.

Second step. The body of the output file is produced. Opening and closing tags, attributes and attribute values are encoded in the order as they appear in the input document (homomorphism). ***An opening tag*** T (or the beginning of an attribute) is encoded with RAE. We follow the same approach of XPRESS [10, Section 4.2] to encode values that represent tags or sequences of tags in a path (see also Section 3.1). The interval corresponding to the tag path from T to the root is computed; then its minimum limit is represented as a 32-bit floating point value, like in the above case. Finally, only the two central bytes are taken to encode the tag: in addition to the first byte (which is fixed), the last one (which is the least significant byte of the mantissa) is truncated to save space. This intentional loss of precision, as we verified, does not prevent a correct tag reconstruction in decompression phase. ***A closing tag*** and the end of an attribute is encoded by the two central bytes of the 32-bit float representation of 1.0070 and 1.00444, respectively. These two values were chosen because truncating does not cause loss of precision. ***An attribute value*** is encoded as follows: if it was encoded in the first step, it is replaced by its 1-byte code followed by the delimiter $\texttt{fe}_h$, otherwise the ASCII string is pasted to output, followed by the delimiter $\texttt{ff}_h$.

4 Experiments

Experimental tests were performed in order to assess benefits and drawbacks of different compression approaches for semantic annotations in the SWoT. Table 1 reports a summary of the most relevant characteristics of compared algorithms. The latest version of DIGCompressor (which is optimized for execution speed) has been used. DIGC1 and DIGC2 respectively refer to the first step only and to both first and second steps of DIGCompressor. DPDT and XPRESS cannot be tested, since they are not publicly available. Only DIGC1, DIGC2 and COX are homomorphic.

Performance comparison has been carried out evaluating three fundamental parameters pointed out in Section 1: **(1)** *compression rate*, defined as $(1 - \frac{size_{compressed}}{size_{uncompressed}})$; **(2)** *turnaround time, i.e.,* the overall duration of the compression; **(3)** *memory usage* of compressor process. Tests were performed using a laptop PC equipped with an Intel P8400 Core 2 Duo CPU (2.27 GHz clock frequency), 3 GB RAM at 800 MHz and Ubuntu 9.04 GNU/Linux operating system

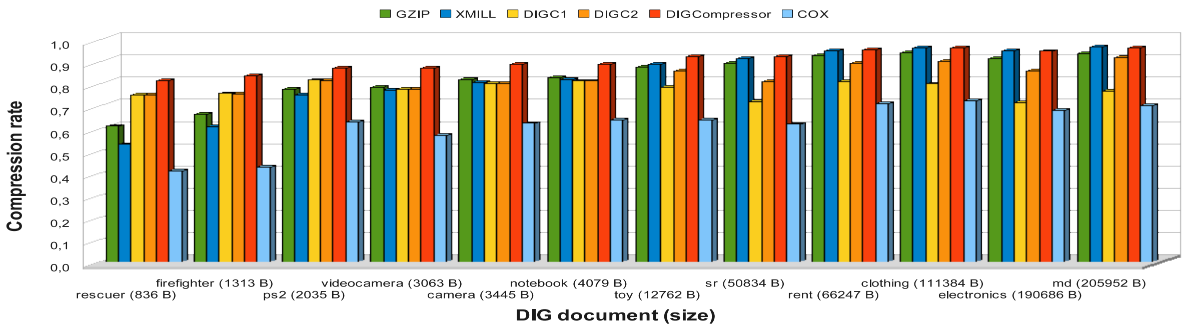

Figure 3. Compression rate

Name	Version	Target	Statistics	Homo-morphic
gzip	1.3.12	Any	Adaptive	No
XMill	0.8	XML	Adaptive	No
DIGCompressor	2.0	DIG	Semi-adaptive	No
DIGC1	2.0	DIG	Static	Yes
DIGC2	2.0	DIG	Semi-adaptive	Yes
COX	1.0	XML	Semi-adaptive	Yes

Table 1. Summary of tested algorithms

with 2.6.27 kernel version and *Valgrind* [11] 3.4.1 profiling toolkit. To obtain a complete picture of performance, 12 DIG documents of different size were used: 6 are semantic annotations of individual objects/products in hypothetical SWoT scenarios and 6 are domain ontologies, which are significantly longer.

Figure 3 shows the resulting compression rate. For each DIG file, the original size in byte is reported. COX and DIGCompressor were respectively the worst and the best performer. Compression rate of COX is somewhat sacrificed to allow general-purpose compression of XML-based Semantic Web languages with support for direct queries on the encoded format. DIGC1 and DIGC2 performed equally or better than gzip and XMill for short annotations, whereas their performance decreased for ontologies.

W.r.t. turnaround time, each test was run 10 times consecutively, and the average of the last 8 runs was taken. Results for annotations are reported in Figure 4-a. DIGC1 and DIGC2 performed very well on such small DIG documents, equaling the highly-optimized gzip tool. XMill and COX suffer from the inherent complexity of their algorithms, but absolute results they produce are still acceptable. For ontology documents, as depicted in Figure 4-b, the situation changed. DIGC2 and DIGCompressor have significantly higher turnaround times for input longer that 100 kB. COX provides acceptable performance, whereas DIGC1 proves itself highly efficient.

Finally, memory usage analysis was performed using *Massif* tool of *Valgrind* debugging and profiling toolkit. For our comparison, only the memory peak was considered. Results are reported in Figure 5. XMill and gzip, due to their particular structure, use almost constant memory amounts regardless of the input size. DIGCompressor exploits about 300 kB for small annotations, growing up to 460 kB for bigger ontologies. COX is much more efficient ($24 \div 35$ kB) for small documents, but its memory peak exceeds DIGCompressor for largest inputs. Finally, DIGC1 and DIGC2 are very lightweight (< 8 kB) for the six annotations, and grow linearly for longer documents while remaining more efficient than the full DIGCompressor.

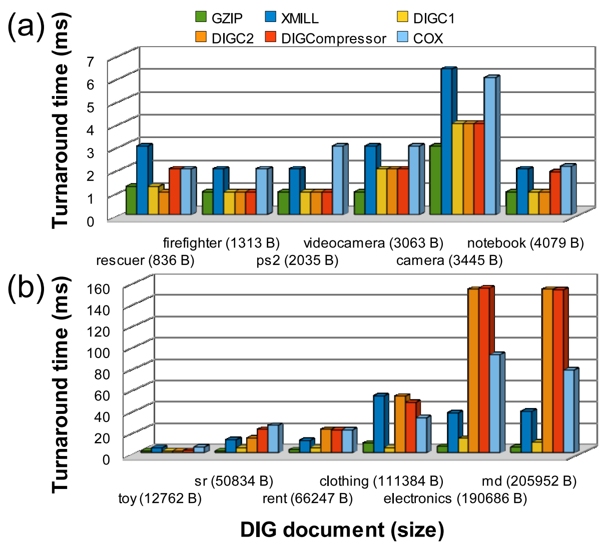

Figure 4. Turnaround time - (a) instance annotations, (b) ontologies

5 Conclusion

Compression techniques are fundamental in scenarios involving mobile devices such as RFID tags or wireless sensors. The paper proposed significant issues in this field, also

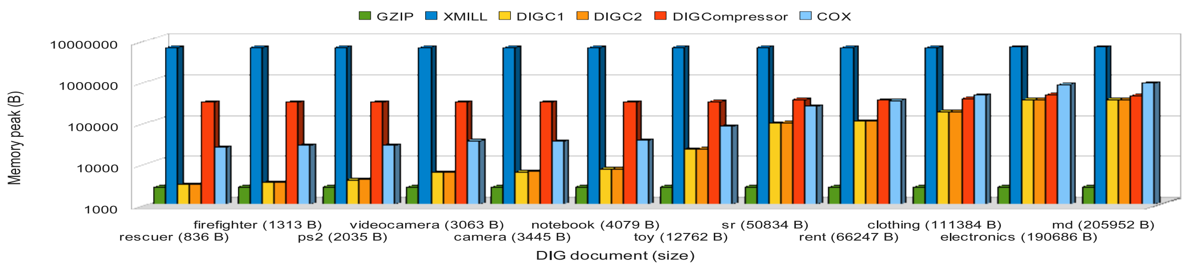

Figure 5. Main memory peak

showing possible future directions: a homomorphic compression approach is outlined allowing to avoid expensive decompressions before querying managed data.

References

[1] S. Bechhofer, R. Möller, and P. Crowther. The DIG Description Logic Interface. In *Proceedings of the 16th International Workshop on Description Logics (DL'03)*, volume 81 of *CEUR Workshop Proceedings*, Rome, Italy, September 2003.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):28–37, 2001.

[3] T. Di Noia, E. Di Sciascio, F. M. Donini, M. Ruta, F. Scioscia, and E. Tinelli. Semantic-based bluetooth-rfid interaction for advanced resource discovery in pervasive contexts. *International Journal on Semantic Web and Information Systems*, 4(1):50–74, 2008.

[4] S. Harrusi, A. Averbuch, and A. Yehudai. XML syntax conscious compression. In *Data Compression Conference (DCC2006)*, March 2006.

[5] P. Howard and J. Vitter. Analysis of arithmetic coding for data compression. In *Data Compression Conference, 1991. DCC'91.*, pages 3–12, 1991.

[6] D. Huffman. A method for the Construction of Minimum Redundancy Codes. In *Proceedings of the IRE*, volume 40, pages 1098–1101, 1952.

[7] ITU. Internet Reports 2005: The Internet of Things, November 2005.

[8] H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. *SIGMOD Rec.*, 29(2):153–164, 2000.

[9] M. Mahoney. Adaptive Weighing of Context Models for Lossless Data Compression. Technical report, Florida Tech. Technical Report CS-2005-16, 2005.

[10] J. Min, M. Park, and C. Chung. A compressor for effective archiving, retrieval, and updating of XML documents. *ACM Transactions on Internet Technology (TOIT)*, 6(3):223–258, 2006.

[11] N. Nethercote and J. Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. In *Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation - PLDI 07*, pages 89–100. ACM Press New York, NY, USA, June 2007.

[12] M. Ruta, T. Di Noia, E. Di Sciascio, F. Scioscia, and E. Tinelli. A ubiquitous knowledge-based system to enable RFID object discovery in smart environments. In *Proceedings of the 2nd International Workshop on RFID Technology - Concepts, Applications, Challenges*, pages 87–100. INSTICC Press, 2008.

[13] P. Tolani and J. Haritsa. XGRIND: A Query-friendly XML Compressor. In *Proceedings of the 18th International Conference on Data Engineering (ICDE.02)*, page 225234. IEEE, 2002.

[14] I. Witten, R. Neal, and J. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[15] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.