Automatic Extraction of Semantic Information from XML Documents

C. N. Wordu & E. O. Bennett
Department of Computer Science,
Rivers State University,
Port Harcourt,
Nigeria
bennett.okoni@ust.edu.ng, chima.wordu@ust.edu.ng

Abstract
The versatility and flexibility of XML documents, although being its hallmark, becomes a problem when it comes to semantics, such that a typical XML file has no obvious semantic information within itself. The semantics of any data source is important to ensure proper data interchange between parties and development of systems which are based on those data. This research project looks at automatically extracting semantic information from XML documents by devising a system to derive XML schema definitions or XSDs internally from these documents and as such contains semantic information about the document and then displays extract content. Java programming language was used for the implementation of the system. The extraction tool makes use of a JAXB based XML processor, which extracts XSD into Java objects internally from XML data. Experiments were conducted on the JAXB architecture against other XML-specific parsing implementations like SAX and DOM. The JAXB, SAX and DOM implementations were tested using XML dataset of varying sizes and it was seen that JAXB was the more efficient XML processor for Java applications overall in terms of unmarhsalling time and coding effort.

Keywords: *XML, information extraction, semantics; WEKA*

1. Introduction

Extensible Markup Language (XML), soon after its conception, was considered to be the future technology of the Web [1]. The semi structured format of the language allows for a wide array of document storage options and applications. Indeed, the main design goals of XML emphasize features such as simplicity, generality, and usability in its use across the Internet [2]. XML data sources exhibit different structures and contents. For effective exchange of XML-based data (and other forms of data as well), the format (i.e., elements, attributes and their semantics) must be fixed.

Although XML has become the common markup language used for the interchange of data over the Internet, XML use has its challenges. One major problem with XML is that the non-specification of semantic information in XML by default could make it difficult for applications to extract such data on demand in an efficient manner simply put, a typical XML file specifies structure and content, but not semantics.

The aim of this paper is to develop a system that wills identity semantics in an XML document and automatically extract the semantics from that document. This will be achieved by developing the tools that would determine the semantics in the structure and content in a selected XML document and test the system using an XML dataset to evaluate its effectiveness in extracting semantic information automatically from the data source.

2. Review of Related Work

Extraction of semantic data from web documents is a relatively new, but vigorously researched field, both in XML documents, database management systems and in computer science in general. Methods developed for semantic information extraction have involved natural language analysis and processing, and data mining of web documents (and XML documents in particular).

Information Extraction in itself is a very general concept, but at the same time is one of the principle applications of computer technology. In one survey of information extracting methods [1], information extraction (IE) is considered to be a process which is a step above information retrieval (IR), which is the locating of relevant documents for a given query. IE is the extracting of facts from relevant sources of data or documents [1].

In [1], approaches for automatic markup of macro and micro structures within rather unstructured XML documents were surveyed and it was stated that whereas macro-level markup is mainly based on information about the layout of a document, micro-level markup typically requires basic linguistic procedures in combination with application-specific knowledge. Authors of the paper said the task of information extraction can be considered as a problem of template filling and stated that a template form has to be defined, and the system is supposed to fill it. The paper presented the Vasari Project, which offers a language and a set of tools for the purpose of micro-level markup of Optical Character Recognition (OCR) scanned texts under the knowledge engineering approach. The project aimed at developing a Web portal providing comprehensive knowledge about art, artists and works of art, using a Vasari Language tool (VasLa) and a Vasari Language Extractor tool (VasLaEx).

Similar to the last paper, a project on the automatic extraction of semantics from law document texts [4] looked at the problem of automatically enriching legal texts with semantic annotation, to effectively index and retrieve legal documents. The researchers developed a computational system for automated semantic annotation of (Italian) law texts
Another research on extracting and modeling the semantic information content of web documents [6] devised an approach meant to assist in extracting and modeling the semantic information content of HTML documents, again, using natural language analysis techniques and a domain specific ontology. The method involved a user tool that gradually extracts and constructs the semantic document model which is represented as XML, with these models representing each document being then integrated to form a global semantic model which would supposedly provide users with a global knowledge model of some domains.

Most of the methodologies investigated in this research use some form of ontology as a knowledge base. In a paper on ontology learning by analyzing xml document structure and content [7] most existing methods for ontology learning from textual documents rely on natural language analysis. The research extended these ontology learning approaches by taking into account the document structure which bears additional knowledge. The documents that dealt with were XML specifications of databases.
SIEU (Semantic Information Extraction in University Domain) which is a semantic search engine developed [8] and confined to the university domain. SIEU uses ontology as a knowledgebase for the information retrieval process.

YaSemIR [9], is a free open-source Semantic Information Retrieval system based on Lucene. It takes one or more ontologies in OWL format and a terminology associated to each ontology in SKOS format to index semantically a text collection. The terminology is used to

annotate concepts in documents, while the ontology is used to exploit the taxonomic information in order to expand these with their subsumers.

The application of automatic extraction of semantics has not only been carried out on electronic data sources but also on manual or paper archives as well. Research in two articles [10] and [11] reported that the application of semantic-based indexing techniques during the transformation of paper documents to electronic format can be used for effective daily processing of large amounts of paper documents in office environments. Both articles made use of a combination of XML and knowledge technologies. This combination is to enable XML distinguish between data, its structure and semantics, allowing the exchange of data elements that carry descriptions of their meaning, usage and relationship. Moreover, the research by [10] and [11] claimed that the combination with XSLT enables browsers to render the original layout structure of the paper documents accurately.

XStruct [3] is an' automatic technique for XML Schema extraction. XStruct extracts a schema for XML data by applying several heuristics to deduce regular expressions that are 1-unambiguous and describe each element's contents correctly but generalized to a certain degree. The presenters of the technique reported that XStruct scales to very large documents (beyond 1GB) both in time and memory consumption; it is able to extract a schema for multiple documents as well as detect data types and attributes.

3. Methodology and Design

Constructive research methodology was chosen for this dissertation because it best fits the criteria for developing the system required to achieve the objectives of this research work. Since the research thus far has shown that there is generally no automatic tool or method that extracts semantic data from XML files, the constructive methodology will provide the steps necessary to develop the new system.

The design methodology to be adopted in this research is an Object Oriented Design. Object oriented design best fits this research because of the nature of XML data, being made up of structured tags representing elements.
In considering the architecture to adopt in this project there are some factors to be considered such as speed, memory usage, ease of programming and the need to work on different platforms. Figure 1 shows the system architecture.

Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of web services. JAXB leverages the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming. JAXB provides the xjc schema compiler tool and the schemagen schema generator tool to transform between XML schema and Java classes.

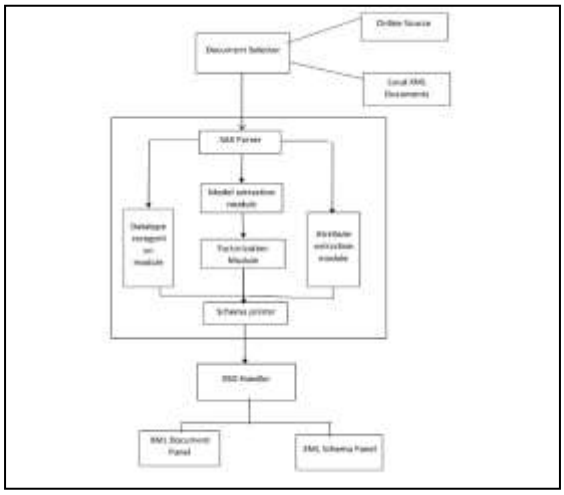

Figure 1: System Architecture

JAXB is an XML-to-Java binding technology that supports transformation between schema and Java objects and between XML instance documents and Java object instances. JAXB consists of a runtime application programming interface (API) and accompanying tools that simplify access to XML documents. JAXB also helps to build XML documents that both conform and validate to the XML schema. Java API for XML-Based Web Services (JAX-WS) leverages the JAXB API and tools as the binding technology for mappings between Java objects and XML documents. JAX-WS tooling relies on JAXB tooling for default data binding for two-way mappings between Java objects and XML documents.
Detailed design handles the implementation section of a system and its sub-systems which determines the logical structure of particular modules and their interfaces to interact with other modules. The detailed design of the system is shown in figure 2.

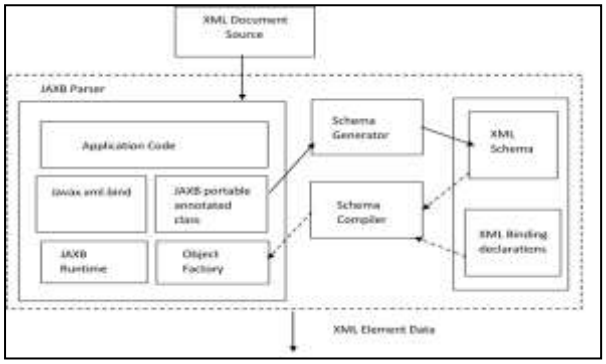

Figure 2: Detailed Design

A JAXB implementation consists of the following architectural components:
i. Schema compiler: Binds a source schema to a set of schema-derived program elements. The binding is described by an XML-based binding language.
ii. Schema generator: Maps a set of existing program elements to a derived schema. The mapping is described by program annotations.
iii. Binding runtime framework: Provides unmarshalling (reading) and marshalling (writing) operations for accessing, manipulating, and validating XML content using either schema-derived or existing program elements.

The general steps in the JAXB data binding process are:
i. Generate classes: An XML schema is used as input to the JAXB binding compiler to

generate JAXB classes based on that schema.

ii. Compile classes: All of the generated classes, source files, and application code must be compiled.

iii. Unmarshal: XML documents written according to the constraints in the source schema are unmarshalled by the JAXB binding framework. Note that JAXB also supports unmarshalling XML data from sources other than files/documents, such as DOM nodes, string buffers, SAX Sources, and so forth.

iv. Generate content tree: The unmarshalling process generates a content tree of data objects instantiated from the generated JAXB classes; this content tree represents the structure and content of the source XML documents.

v. Process content: The client application can modify the XML data represented by the Java content tree by means of interfaces generated by the binding compiler.

vi. Marshal: The processed content tree is marshalled out to one or more XML output documents. The content may be validated before marshaling

4. Experimental Results

The system was implemented using Java 7. Experiments were run on an Intel® Pentium® dual CPU T3200 @ 2.00GHz machine with 4.00GB memory.

Testing was carried out on the system's XML parsing operations. This was done using three different datasets consisting of XML documents containing a list of Employee entities. The unmarshalling modules of the system has two Java objects to extract the XML data, one object to represent the Employee entity in the XML document and the other object to represent the collection of employees or employee list.

Figures 3, r and 5 show an XML instance document with the Employee entity, and the corresponding Java objects.

The objective of the testing was to get the entities in the XML document to the corresponding Java objects. This unmarshalling operation is performed by the JAXB, DOM and SAX implementations using the Java objects in their own separate tests.

For each of the parsing implementations (JAXB, DOM & SAX) tests were run five times for three files which contain a collection of Employee entities. The first file contained 100 Employee entities and was five kilobytes in size. The second contained 10,000 entities and was 500 KB in size and the third contained 250,000 Employee entities and was 15 Megabytes in size.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<employee>
    <emp id="100">
        <firstName>John</firstName>
        <income>2000.15</income>
        <lastName>Abraham</lastName>
    </emp>
    <emp id="200">
        <firstName>Peter</firstName>
        <income>2500.02</income>
        <lastName>Isaac</lastName>
    </emp>
    .......
```

Figure 3: XML document with Employee entities

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "id",
    "name"
})
public class Person {
    private String id;
    private String name;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String value) {
        this.name = value;
    }
}
```

Figure 4: Java object for Employee entities

```
@XmlRootElement(name = "employee")
@XmlAccessorType(value = XmlAccessType.FIELD)
public class Employee {

    @XmlElement(name = "emp")
    private List<Emp> employees;

    public Employee() {
       return employees;
    }

    public List<Emp> getEmployees() {
        return employees;
    }

    public void setEmployees(List<Emp> employees) {
       this.employees = Emp;
    }
}
```

Figure 5: Employees Java object

Table 1 shows the average unmarshalling times for the JAXB, DOM and SAX parser implementations.

Table 1 Average Times for JAXB, DOM & SAX

XML Parser	100 Employees time (ms)	10000 Employees time (ms)	250000 Employees time (ms)
JAXB	14.4	54.8	1390.6
DOM	4.6	59	1916.2
SAX	1.8	27.2	630.8

5. Discussion of Results

Figures 6, 7 and 8 show the average unmarshalling times of the three XML parsers for the 100, 10000 and 250000 Employees entities respectively.

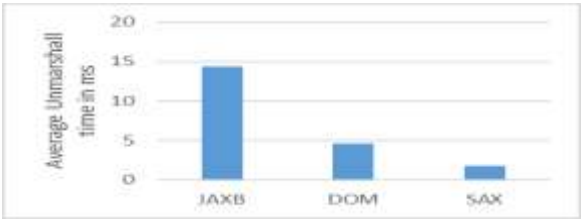

Figure 6: 100 Employees entity average unmarshalling time (ms)

Figure 6 suggests that JAXB had the highest unmarshall time for the 100 Employees entity XML dataset while SAX had the least. It may seem that SAX is the best at this point but it should be noted that SAX implementation requires more coding in the marshalling process than the other two XML parsers, and JAXB requires the least coding, meaning JAXB us

more efficient for this smaller dataset.

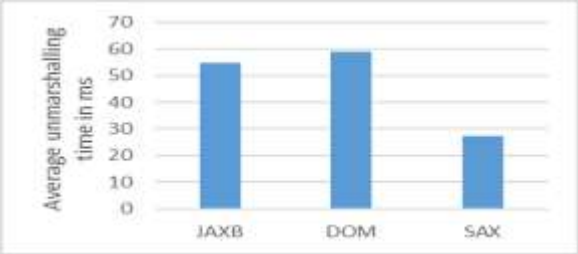

Figure 7: 10000 Employees entity unmarshalling average time (ms)

Figure 7: shows that JAXB average unmarshalling time is lower than DOM. Clearly, as the size of the dataset increases the JAXB average unmarshalling time improves, although SAX unmarshalling time is lower.

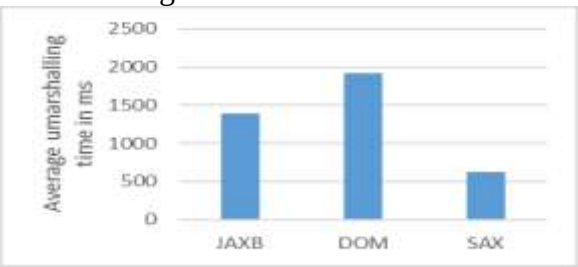

Figure 8 : 250,000 Employees entity unmarshalling average time (ms)

Figure 8 shows the improvement of JAXB average unmarshalling time over DOM. SAX is still lower though, showing that SAX is ideal for very large XML datasets. But for programming or coding efficiency, JAXB still remains a suitable alternative in XML parsing overall.

This research work was to devise automatic methods of extracting semantic information from XML documents. In the process, data mining techniques were used to derive XML Schema Definitions or XSDs form XML documents automatically. The system that was developed used tools based on three XML processor types: JAXB, DOM and SAX. These XML processors or parsers are the major components involved with the extraction of semantic information form XML files. A system was developed using Java to test the performance of the JAXB, DOM and SAX API implementations. The performance of the parsers was tested in terms of time taken to read XML elements form the documents into the Java objects thus deriving internally the XSD information from the XML document, a process called marshalling. The results showed that the throughput of the JAXB implementation was more efficient that the SAX and DOM parser implementations in terms of coding effort and umarshalling time. The tests were done using three XML datasets of different sizes but similar structural features, and although SAX was faster at marshalling larger datasets, JAXB was more efficient in terms of coding for the operation and had a faster marshalling time than DOM.

In conclusion, JAXB, SAX and DOM are all efficient for parsing XML data in Java applications. Although SAX seems to be the faster XML processor when parsing very large XNL documents.

6. Future Works

Based on the efficiency of the methods devised in this research, it is recommended that JAXB and SAX based XML processors be used for more efficient parsing to extract semantic data

from XML documents

References

Abolhassani, M. Fuhr, N. and G¨overt, N. (2003), *Information Extraction and Automatic Markup for XML Documents*

Antonacopoulos, A. and Karatzas, D. (2005), Semantics-Based Content Extraction in Typewritten Historical Documents, *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR2005), Seoul, Korea, IEEE*, 48-53

!Aussenac-Gilles, K. and Kamel, M. (2006), Ontology Learning by Analyzing XML Document Structure and Content, 1-15.

Azman Noah, S., Zakaria, L. &Alhadi, A. C., (2009). Extracting and Modeling the Semantic Information Content of Web Documents to Support Semantic Document Retrieval, 2009

Biagioli, C., Francesconi, E., Spinosa, P., Taddei, M. (2003), The NIR project. Standards and tools for legislative drafting and legal document Web, Proceedings of the International Conference of Artificial Intelligence and Law, Edinburgh.

Bohring, H. and Auer, S. (2005), Mapping XML to OWL Ontologies, Leipziger Informatik-Tage – LNI, 72 (1), 1-10

Buscaldi, D. and Zargayouna, H. (2013), YaSemIR: Yet another Semantic Information Retrieval System, ESAIR '13 Proceedings of the sixth international workshop on Exploiting, 13-16. Databases, 61-66.

Fallside, D.C and Walmsley, P. (2004) XML Schema Part 0: Primer. W3C recommendation, http://www.w3.org/TR/xmlschema-0/.

Fennell, P. (2013), Extremes of XML. A session at XML London 2013.

Hegewald, J., Naumann, F., and Weis, M. (2006), XStruct: Efficient Schema Extraction from Multiple and Large XML Documents, 2nd International Conference on Data Engineering Workshops (ICDEW'06), 1 (81), 1-10.

Rajasurya S., Muralidharan T., Devi, S. (2012), Semantic Information Retrieval Using Ontology in University Domain, International Journal of Web & Semantic Technology, 3 (4), 55-67.

Soria, C., Bartolin, R., Lenc, A., Montemagn, S. and Pirrell, V. (2005), Automatic extraction of semantics in law documents, Proceedings of the 10th international conference on Artificial intelligence and law, 253-266.