XAPP: An Implementation of SAX-Based Method for Mapping XML Document to and from a Relational Database

Yetunde Akinwumi
Department of Computer Science
Redeemer's University, Nigeria
Akinyetty16@gmail.com

Joshua Ayeni
Department of Computer Science, Redeemer's University, Nigeria
ayenij@run.edu.ng

Samson Arekete
Department of Computer Science, Redeemer's University, Nigeria
areketes@run.edu.ng

Mba Odim
Department of Computer Science, Redeemer's University, Nigeria
odimm@run.edu.ng

Adewale Ogunde
Department of Computer Science
Redeemer's University, Nigeria
ogundea@run.edu.ng

Bosede Oguntunde
Department of Computer Science, Redeemer's University, Nigeria
oguntunden@run.edu.ng

Abstract: *Extensible Markup Language (XML) is the standard medium for data exchange among businesses over the Internet, hence the need for effective management. However, since XML was not designed for storage and retrieval, its management has become an open research area in the database community. Existing mapping techniques for XML-to-relational database adopt either the structural mapping or the model mapping. Though numerous mapping approaches have been developed, mapping and reconstruction time had been problematic, especially when the document size is large and can hardly fit into main memory. In this research, an application codenamed XAPP, a new lightweight application that adopts a novel model mapping approach was developed using Simple API for XML (SAX) parser. XAPP accepts a document with or without Document Type Definition (DTD). It implements two algorithms: one maps XML data to a relational database and improves mapping time, and the other reconstructs an XML document from a relational database to improve reconstruction time and minimise memory usage. The performance of XAPP was analysed and compared with the Document Object Model (DOM) algorithm. XAPP proves to perform significantly better than the DOM-based algorithm in terms of mapping and reconstruction time, and memory efficiency. The correctness of XAPP was also verified.*

Keywords: *Extensible markup language, XML document, relational database, reconstruction time, mapping time.*

Received July 16, 2020; accepted October 21, 2021
https://doi.org/10.34028/iajit/19/4/2

1. Introduction

In this information era, the World Wide Web (WWW) has become an important medium used for a wide range of activities that include e-banking, e-learning, e-mail, e-commerce, e-library, etc. These Internet-based activities lead to the generation of large amounts of data [4]. Researchers and database vendors have encountered difficulties in exchanging a large amount of data between organisations using Extensible Markup Language (XML) technology due to variations in data formats used to describe data [3, 5].

XML has largely become a de facto medium for data exchange and representation on the Internet due to its flexibility and self-describing nature [1, 9]. It provides a generalised structure for data interchange regardless of the platforms and data models of the applications. XML enables communications between different computing systems, and its openness allows data exchange between virtually any hardware, software and operating systems [9]. A multimodal architecture combining Voice XML and InkXML to develop a multimodal voice and ink mobile applications for man-machine communication has been proposed in [10]. Themulti modal interface architecture is designed to broaden the spectrum of general users. The integration of Voice XML and InkXML would provide a standard data format to facilitate Web-based development and content delivery, enabling diverse applications ranging from complex data entry and text editing applications to Web transactions to be implemented on the system.

Most semi-structured data on the web are represented in XML, hence the need for effective storage and retrieval of XML data [3]. Mapping XML documents to Relational Databases (RDB) has been studied because it entails features such as data integrity, crash recovery and multi-user access, which are not directly available in XML technology.

Researchers have used two broad models in mapping XML to a RBD, namely: model-based and

structural-based mappings. In model-based mapping, an XML document with or without Document Type Definition (DTD) or XML schema can be stored in a relational schema since it requires a fixed RDB scheme. In structural-based mapping, the XML documents are stored in the RDB scheme based on the structure, that is, DTD or XML schema stated in the document. Considerable research efforts have been focused on model-based approach because it is considered most suitable for storing and querying XML documents [3].

Several model-based mapping approaches have been proposed to store and reconstruct XML data into and from the relational database, but these approaches have been carried out using Document Object Model (DOM) parser, which requires a lot of time, especially when the document size is large. This current study focuses on model mapping approach which accepts XML with or without DTD to map XML data to and from a relational database using SAX parser to reduce the processing time.

1.1. Related Works

An approach for the storage and retrieval of XML documents using relational databases was made by Seif *et al*. [8]. The approach enables XML documents to be stored using a fixed relational schema without any information of XML schema or DTD. It used two algorithms: XR and RX, with XR converting XML data to relational data, and RX extracting data from a database and insert them into an XML document. The relational schema consists of three tables-document table, element table and attribute table. The research used a DOM parser which produces large data for anything beyond a small size document, hence, memory inefficient for large documents.

Qtaish and Ahmad [7] presented a mapping done on only distinct ancestor paths for all leaf nodes of the XML tree into its RDB with the inner nodes ignored to minimise the storage space. The mapping comprises of two algorithms: X to DB and X to Structured Query Language (SQL). X to DBmaps XML documents to a fixed RDB using a DOM parser to decompose the XML document into a predefined RDB scheme. X to Translates XPath queries into the corresponding SQL queries based on the Ancestor_Path and Leaf_Node tables to achieve a shorter response time. The reconstruction of an XML document from the relational database is imperfect because not all nodes are stored; the model was targeted at conserving storage space and query response time.

Atay *et al*. [2] proposed a mapping for storing XML data into a relational database using schema mapping and data mapping. The schema mapping takes an XML DTD as input and generates a relational schema with the Open Document Text Document (ODTD) Map algorithm, which consists of three steps: simplifying XML DTD to reduce complexity, generation of DTD graphs for preparation of the mapping and using the DTD graphs for the creation of the relational schema. After the creation of relational schema, the next step is to insert data from XML document into the generated tables, a process known as data mapping. The relational schema consists of six tables. The limitation is that too many tables are generated, each element from the in-lined DTD graph has its own table, making reconstruction of XML documents difficult.

Zhu *et al*. [12] presented a path-based model mapping approach, which identifies the path among the non-leaf nodes. The mapping generates two tables to store the nodes: Path Table and Leaf Table. To construct the tables, the position information is marked-up with labelling scheme of (Level, [P-path ID, S-order]), where Level represents the depth of the current leaf node in the XML tree, P-pathID stands for the path id of the direct parent node, and S-order depicts the position number of the current leaf node in the direct parent node. The information serves to identify the complex node relationship. The approach was reported to be experimentally better than the s-XML with regards to storage space and storage time. The limitation is that reconstruction of an XML document from the relational database is difficult since not all nodes were stored.

1.2. An Overview of XML Document

XML documents contain a large amount of data structured hierarchically. They are composed of elements known as the main building blocks that are tagged. The elements are structured hierarchically from the outermost element called the root element, followed by the child element or nested element [11]. Each XML document is composed of two parts: the prologue and the document element (also known as the root element). The prologue always comes first, before the root element on top the document. The prologue consists of the XML declaration, DTD, comments, processing instructions and whitespace while the document element or root element comes immediately after the prologue containing the content of the document. The document element consists of one or more elements embedded in one root. It consists of opening tag root element, child element and content and closing tag root element [7]. Figure 1 depicts an example of an XML document.

There are two types of XML documents, the well-formed and valid documents. Well-formed XML documents are documents without DTD and XML schema which follow the rules of XML syntax, such rules are:

a. The document must contain one root element.
b. The document must contain one or more elements.
c. Every element must have a start tag and end tag.

d. Every element must be nested correctly without overlapping.
e. Every attribute values must be quoted.

Valid XML documents, on the other hand, are well-formed documents that do not only conform to syntax rules but also conform to schema rules. Schema can be defined as the grammar defining the logical structure of XML document in either DTD or XML schema (XML Language Schema).

```
<?xml version= "1.0" encoding= "UTF-8" standalone= "no"?>
<!DOCTYPE document system "person.dtd">
<!-- Here is a comment -->
<?xml-stylesheet type="text/css" href="Styles.css"?>
<person>
<person born="26/12/1791" died="18/10/1871">
<name>
        <first_name>Charles</first_name>
        <last_name>Babbage</last_name>
</name>
<occupation>Philosopher</occupation>
<occupation>Mathematician</occupation>
<occupation>Inventor</occupation>
</person>
</person>
```

Figure 1. Example of XML document.

2. Material and Method

2.1. The XAPP SAX Based Approach

XAPP uses simple API for XML (SAX) parser [8] for parsing XML document to store it in a relational database. SAX parser is used in place of DOM parser to deal with large XML documents. It parses XML document as a sequence of events (i.e., start Document, end Document, start Element, end Element, etc.,). This approach contrasts DOM parser, which constructs the whole document tree (in memory) first and then parses it. The advantage of DOM parser over SAX parser, however, is that it offers XML update while the latter provides XML as read-only.

Input data were taken from the Digital Bibliography Library Project (DBLP) for University of Washington Repository [6] and results were generated based on different sizes.

2.2. XAPP Architecture

Figure 3 depicts the XAPP architecture. It provides amapping of an XML document into a relational database and reconstructing relational data back to an XML document. The XML document is passed into the SAX parser facility where a parsed XML document is generated. The resultant document is inserted into the RDB through SQL statement and Java™ Database Connectivity (JDBC) API connectivity. Reconstructing XML from RDB requires that an SQL result set is passed into the XML reconstruction algorithm. The system flowchart is depicted in Figure 2.

2.3. Mapping Algorithm for XML Document to Relational Database

The algorithm takes XML document file as input, maps and stores its contents in a relational database Algorithm (1). The process takes two basic steps. First, data is extracted from XML with SAX parser. To extract data, the XML file is fed as input into the SAX parser, which parses the file by taking one factory instance from SAX Parser Factory using the parse method to parse it.

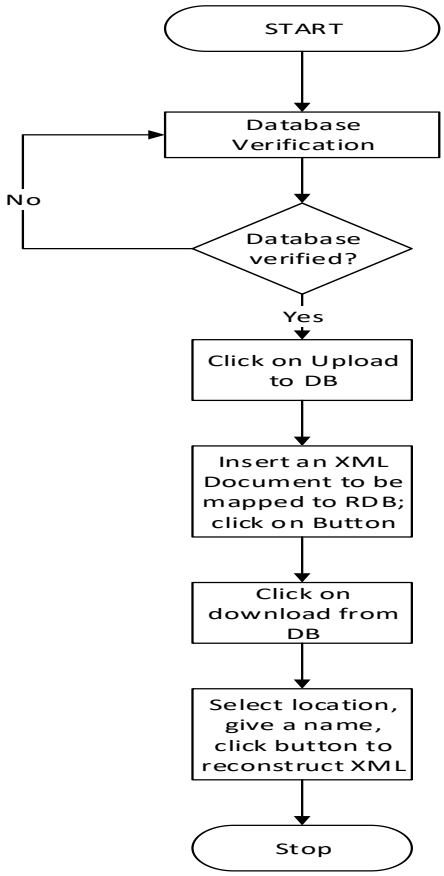

Figure 2. System flowchart of XApp.

SAX parser provides better memory management and faster execution time. In addition, it does not keep any data in memory so it can be used for very large files. When SAX Parser initiate the parsing process, it scans the document and when it encounters the start or end tag it will invoke the corresponding event handling method in the public void startElement() and public void endElement() methods. Document Handler keeps track of callback events and notifies an application program of the data in XML document. The second step is inserting data into MySQL. This process involves using SQL statement to insert XML data into relational database through Java Database Connectivity Application Program Interface (JDBCAPI).

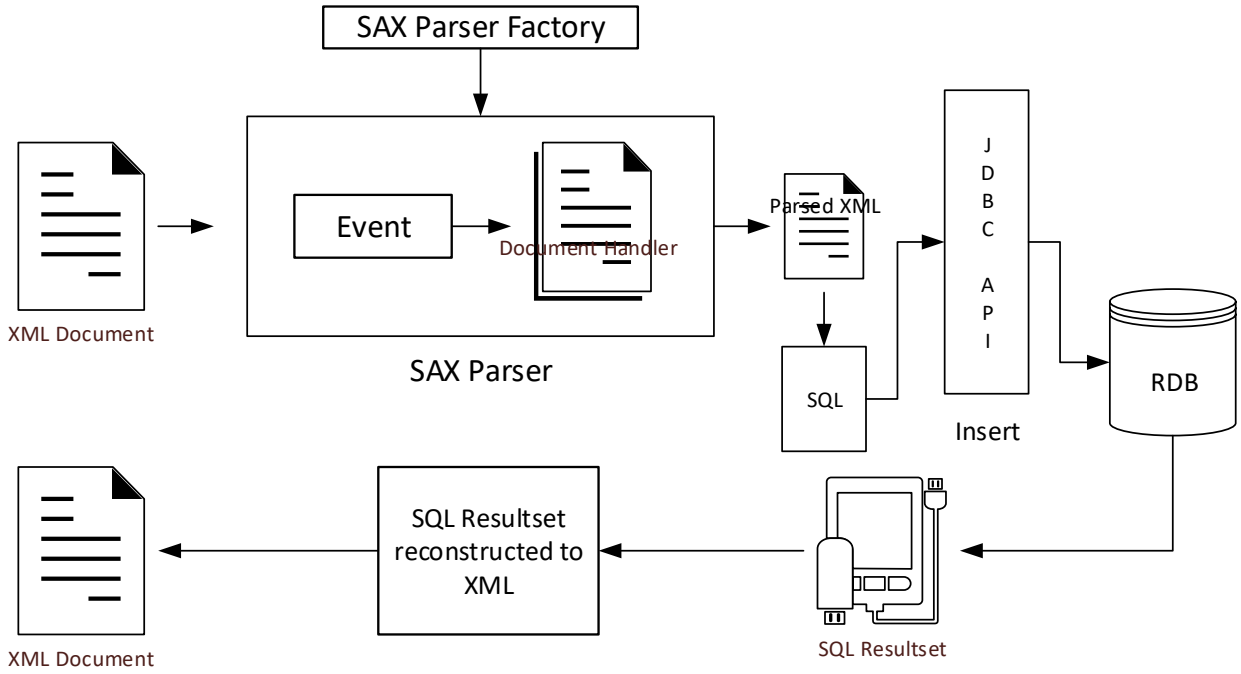

Figure 3. The XAPP architecture.

2.4. Reconstructing Algorithm from Relational Database to XML file

Algorithm (2) describes the extraction of data from a relational database into an XML document. The docBuilder creates a root element of the document "results" from the first tuple of the SQL Result Set and append that element to our document. The result set is then traversed, creating row regular element from each row and append them to the root element results. Looping through all columns, the column names and values are obtained. The node element is created from column name and append value as child node and append node element to regular element row.

The function "get Document As XML" passes the parameter "doc" which is the resultant document from the document builder and returns the document as XML string using the StringWriter() function. StreamResult() function then constructs a result object which is passed into the transform function object along with the DOM source object created from the document. Transformer object is created from Transformer Factory object and set Output Property defines necessary output parameters of transformer, specifying the output format as XML and using the encoding standard "ISO-8859-1" to support all characters in the document.

2.5. Relational Schema

The relational schema design for the SAX-based approach is based on the elements the XML document entails.

3. Result and Discussion

The two algorithms for mapping between an XML document and relational database were implemented, and a series of performance experiments were carried out and compared with a DOM based algorithm. The experiments were done to check the effectiveness of our new approach. The implementation process and experimental results were captured and documented.

All experiments were executed on an Intel(R) Core (TM) i3-2328M System, CPU of 2.20 GHz, 4GB RAM and 445GB HDD and Windows 8 64-Bit OS. Implementation tools were PHP 5.6.25, MySQL server 5.7.14 and Apache server 2.4.23.

Performance metrics were based on the time taken to map XML document data to RDB and time to reconstruct XML document from RDB. The DOM-based algorithms used for comparison was adopted from the study by [12].

Tables 1 and 2 depict the mapping and reconstruction time from an XML document to RDB and back. The SAX-based algorithms outperform the DOM-based algorithms in terms of execution time because DOM creates a tree structure of the XML document before parsing takes place but this is not required in SAX-based algorithms. The outputs of the SAX-based algorithms are therefore more memory efficient for the same reason that tree structures are not created. Furthermore, it was observed that while mapping time from XML to RBD was close for both SAX-based and DOM-based algorithms (ranging from 72% to 57% as we move from 400KB to 1MB, the reconstruction time for SAX-based algorithm was much shorter (from 0.5% to 0.2% as we move from

400K to 1MB), implying that it requires a lot of time to generate the tree structures.

Algorithm 1: XML to RDB

Input: XML Document
Output: XML data stored in RDB

```
Function XMLParser(){
SAXParserFactoryspfac = SAX ParserFactory.newInstance() //Create a "parser factory" for creating SAX parsers
SAXParsersp = spfac.newSAX Parser() //Use the parser factory to create a SAX Parser object.
MyHandler handler = new MyHandler() //Create an instance of this class; it defines all the handler methods
sp.parse("doc.xml", doc_handler) //Finally, tell the parser to parse the input and notify the handler.
}
Function docHandler(){ // this handles the callback events triggered by the SAX parser
Procedure startElement(String name, AttributeListattrs){ // this method is called every time the parser encounters the beginning of a new element
 For each node Loop {
If (node == ELEMENT_NODE){ //if there is node of element node type
node.getElementName //get element name
node.getElementValue(value) //gets the element value
If (element hasatrribute){ //if element node has attribute node type
node.GetAttributeName(name)//get attribute name
node.GetAttributeValue(value) //gets attribute value
If (element has text values){
node.getTextValues (values) // get text value.
}}}
Procedure characters (){// this method calls for the text contents between the start and end tags of the XML document element
}
Procedure end Element(String name){ // When the parser encounters the end of an element, it calls this method
}}
End loop
INSERT values INTO TABLE
}
```

Algorithm 2: RDB to XML

Input: SQL ResultSet
Output: XML document

```
Connecton= mysql_connect ('local host', 'root', 'username')
DocBuilder(){ //Build XML Document from database. The XML object is returned to main method where it is written to flat file
ResultSetrs= "SELECT *FROM table_name; //perform select query to the get relational data
Document doc = builder.newDocument() //creates a new instance of doc from the Document builder
Element results = doc.createElement("Results") // Creating result root elements within XML doc
doc.append Child(results) //Populating XML doc root element with data by appending results
while (rs.next()) { //iterates through result set
Element row = doc.createElement("Row") //Creating regular element from each row
Results.appendChild(row) //Populating with row data by appending results
for (inti = 1; i<= colCount; i++) { //loop through all columns
String column Name = rsmd.getColumnName(i) //gets column name
Object value = rs.getObject(i); // gets value as object
Element node = doc.createElement(columnName) //create element node from column name
node.appendChild(doc.CreateTextNode(value.ToString())) //populate node with data as string
Row.AppendChild(node) // adding the node element to the regular element
}}}
Function getDocumentAsXml(Document doc) { //function converts newly built document doc above to XML string
Source = new Source(doc) //constructs a source object from the XML doc
Transformer Factorytf = TransformerFactory.new Instance() //declares tf as new instance of transformer from Transformer Factory
Transformer transformer = tf.newTransformer() //creates transformer from with tf instance
Transformer.SetOutputProperty(OutputKeys.METHOD,"xml"); //specifies the output resultformat as xml
Transformer.SetOutputProperty(OutputKeys.ENCODING, "ISO-8859-1") //specifies the encoding
StringWritersw = new StringWriter() //constructs a XML string writer object sw
StreamResult sr = new StreamResult(sw) //constructs a result object sr
Transformer.Transform(domSource, sr) //tells the transformer to operate on the source objectDom Source and output to the result object sr.
 return sw.toString() //returns XML string
}
```

Table 1. Mapping time (sec) for XML to RBD.

Algorithm	Document Size			
	400KB	600KB	800KB	1MB
SAX	2,987.3	4,239.2	4,536.2	5,111.3
DOM	4,133.1	5,804.2	7,074.0	8,948.8

Table 2. Reconstruction time (sec) for RBD to XML.

Algorithm	Document Size			
	400KB	600KB	800KB	1MB
SAX	333.3	410.5	415.2	438.4
DOM	64,695.9	95,185.6	121,258.7	272,405.8

The complexity of this algorithm is of order $\odot(n)$, that is, a linear complexity. In the algorithm, a search is performed using a loop statement. The loop statement "for" and "while" were executed according to the number of elements and number of attributes in the XML document, viz:

$$T_{ea} = n_e + n_a \quad (1)$$

Where n_e is the number ofelements, n_a is the number of attributes and T_{ea} is the total number of elements and attributes of the XML document.

4. Conclusions

Several DOM-based algorithms have been developed. However, the intrinsic problem of DOM-based algorithm is that they consume a lot of time and space. In this study, a new mapping approach was proposed

to minimise the time consumed in mapping from XML to RBD and reconstruct from RBD to XML. The application developed code-named "XAPP" accepts XML document with or without DTD to be mapped into a relational database. The new approach solves the problem of mapping an ordered hierarchical XML to an unordered tabular relational database and enables the use of relational database systems for storing and reconstructing XML data using a SAX parser. The method works efficiently for large XML documents. The two algorithms were implemented and tested, with experiments carried out on four different sizes of DBLP dataset (400, 600, 800, and 1000KB). The mapping and reconstruction times were taken and compared with results from the DOM-based algorithm. The results show that XAPP using SAX-based algorithms perform better than the DOM-based algorithm in terms of mapping and reconstruction speed as well as memory efficiency. The reconstruction time for SAX-based algorithm was much shorter, implying that it requires a lot of time to generate the tree structures. These algorithms would, therefore, be very useful in applications where time and memory are of essence and when the DOM tree structures are not required during reconstruction.

While a Virtual Token Descriptor (VDT) parser has been reported to post a better performance than DOM-parser or even SAX-parser, this has not been considered in this study. It can be an area of investigation in future.

References

[1] Ahmad K., "A Comparative Analysis of Managing XML Data In Relational Database," *in Proceedings of Intelligent Information and Database Systems*, Daegu, pp. 100-108, 2011.

[2] Atay M., Chebotko A., Liu D., Lu S., and Fotouhi F., "Efficient Schema-based XML-to-Relational Data Mapping," *Information Systems*, vol. 32, no. 3, pp. 458-476, 2007.

[3] Bousalem Z. and Cherti I., "XMap: A Novel Approach to Store and Retrieve XML Document in Relational Databases," *Journal of Software*, vol. 10, no. 12, pp. 1389-1401, 2015.

[4] Dwebi I., "Automatic Mapping of XML Documents Into Relational Database," Doctoral Thesis, University of Huddersfield. School of Computing and Engineering, 2010.

[5] Fakharaldien M., Edris K., Zain J., and Sulaiman N., "Mapping Extensible Markup Language Document with Relational Database Management System," *International Journal of Physical Sciences*, vol. 7, no. 25, pp. 4012-4025, 2012.

[6] Miklau G. and Suciu D., XML Data Repository, University of Washington, Retrieved 06 21, 2019, from http://www.cs.washington.edu/research/xmldatasets/, 2003, Last Visited, 2022.

[7] Qtaish A. and Ahmad K., "XAncestor: An Efficient Mapping Approach for Storing and Querying XML Documents in Relational Database Using Path-Based Technique," *Knowledge-Based Systems*, vol. 114, pp. 167-192, 2016.

[8] Seif E., Fath E., and Haj E., "A Labeling DOM-Based Tree Walking Algorithm for Mapping XML Documents into Relational Databases," *PhDThesis*. University of Khartoum, Faculty of Mathematical Sciences. Khartoum: University of Khartoum. Retrieved 10 31, 2018, from http://khartoumspace.uofk.edu/handle/123456789/9129, 2011, Last Visited, 2022.

[9] Subramaniam S., Haw S., and Hoong P., "S-XML: An Efficient Mapping Scheme to Bridge XML and Relational Database," *Knowledge-Based Systems*, vol. 27, pp. 369-380, 2012.

[10] Trabelsi Z., "A Generic Multimodal Architecture for Integrating Voice and Ink XML Formats," *The International Arab Journal of Information Technology*, vol. 1, no. 1, pp. 93-101, 2004.

[11] Wang Q., Ren Z., Dong L., and Sheng Z., "Path-based XML Relational Storage Approach," *Physics Procedia*, vol. 33, pp. 1621-1625, 2012.

[12] Zhu H., Yu H., Fan G., and Sun H., "Mini-XML: An Efficient Mapping Approach between XML and Relational Database," *in Proceedings of IEEE/ACIS 16th International Conference on Computer and Information Science*, Wuhan, pp. 839-843, 2017.

Yetunde Akinwumi holds a BSc degree from AfeBabalola University Ado-Ekiti and MSc degree from Redeemer's University Ede. She currently works on Information Modeling & Management, Data Design, mobile health management and web programming. She is currently a Research Assistant in Computer Science, Redeemer's University Ede.

Joshua Ayeni holds a PhD from the University of Waterloo UK. He is a Professor of Computer Science, University of Lagos and Redeemer's University, Ede Nigeria. He is a Fellow of the Computer Registration Council of Nigeria (CPN). He has authored a book in Computer Science and published many articles in reputable peer reviewed journals. His research interest is in the area of formal modelling and simulation.

Samson Arekete holds a PhD degree in Computer Science. He is a Professor of Computer Science at the Redeemer's University, Ede, Nigeria. His research interests are in Artificial Intelligence, Intelligent Agents Systems and Modelling. He has published articles in learned journals and attended conferences.

Mba Odim is a Reader (Associate Professor) in The Department of Computer Science, Redeemer's University, and Ede. He holds a PhD degree in Computer Science and has published widely both local and internationally. He is a member of the Nigeria Computer Society (NCS) the Computer Registration Council of Nigeria (CPN). He holds the Artificial Intelligence Analyst - Mastery and Explorer Awards of IBM. His research interests include but not limited to Machine Learning, Deep Learning, Data Mining, Database Systems and Information Security.

Adewale Ogunde is a Professor in Computer Science. He holds a PhD degree in Computer Science and is a registered member of the Computer Professionals Registration Council of Nigeria, Nigeria Computer Society, IAENG and several other professional bodies. He has published widely in both local and international journals. His research interests are in the area of Artificial Intelligence: data mining, machine learning, and Big Data with other intelligent and knowledge-based systems.

Bosede Oguntunde holds a PhD degree from the University of Ibadan. Nigeria. She is a Senior Lecturer in Computer Science. Her research interests are in Data Communication, Networking, Bioinformatics and Machine Learning. She has published articles in learned journals and presented papers at conferences.