Arkadiusz Osypiuk
Tomasz Grudniewski

JAVA INTERFACES FOR XML DOCUMENT HANDLING

Key words: Java, XML, Java interfaces, SAX, DOM, JAXP

Introduction

Java is commonly thought to have the widest range of API interfaces for XML handling. It is Java which sets standards for the manner of handling XML from the application level, although such languages as C, C++ and Perl are catching up. Currently, there are three major leading API interfaces for XML handling:

- **SAX** (*Simple API for XML*)
- **DOM** (*Document Object Model*)
- **JAXP** (*JAVA API for XML Parsing*)

Seen from the application point of view, XML document processing can be divided into two stages, shown in Fig. 1. These are: document processing and analysing the data contained in it.
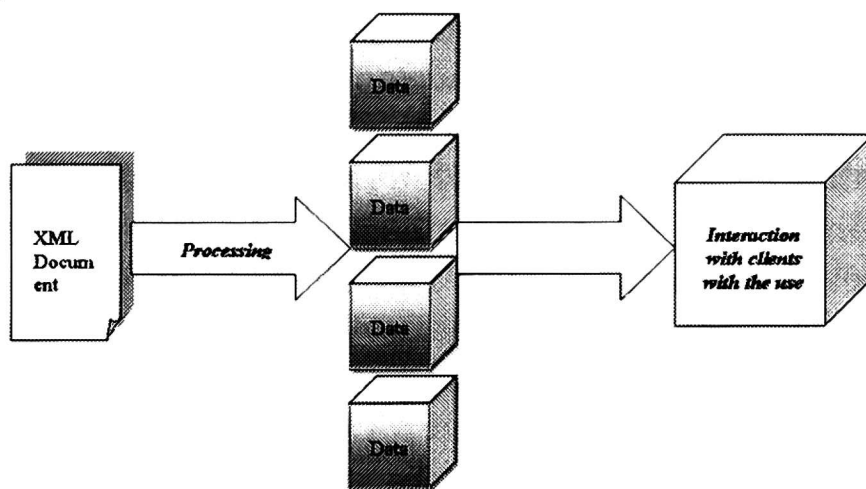

Fig. 1. An XML document processing cycle
from an application point of view

1. SAX (Simple API for XML)

SAX is a simple API interface for XML handling. To process XML data, it employs structure-based event handling. At each stage, events which may

occur are defined. For example, SAX defines the interface *org.xml.sax.ContentHandler*, which in turn defines the methods, such as *startDocument()* and *endElement()*. Such an interface helps gain control over the relevant stages of XML document processing. A similar interface has been created for handling errors and lexical structures. A set of errors and warnings is defined to help handle different situations that may occur when an XML document is being processed, *e.g.* encountering an incorrect or incorrectly formatted document. New behaviours may be added, which help handle tasks associated with highly specific applications – all this within the XML standard application.

1.1. How SAX works

The sequential model offered by SAX does not offer free access to an XML document. When using the SAX, we take information on the XML document when it is done by the parser and – like a parser – we lose the information. When another element appears, information in the previous element is inaccessible because the element has not yet been processed. When the next element appears, we cannot go back to the previous one. Obviously, we can retain information encountered while processing, but it may be difficult to encode such special cases. An alternative could be provided by creating a representation of an XML document in the memory, *i.e.* the DOM model.

1.2. Limitations of SAX

Another task which is difficult to perform with the SAX interface is going from one element to another, situated on the same level. Access to elements through SAX is largely hierarchical and sequential. We get access to the final element of a node, subsequently we go "up" the tree and then we again go down to another element at the bottom of the hierarchy. There is no clear reference to the hierarchy "level" on which we are at the moment. Although levels can be identified by means of sophisticated counters, basically SAX is not suited to such operations. It lacks an implemented concept of a sibling element, the next element on the same level; it is also impossible to check which elements are embedded in which.

2. DOM (Document Object Model)

DOM is an API interface for the Document Object Model. SAX only provides access to data in an XML document, while DOM enables the user to process data. An XML document in a DOM interface is represented as a tree-like structure. Since such a way of representation has been known for long, it is not difficult to search and process such structures from a programming language level. In a DOM interface, the entire XML document is entered in memory and all the data are stored as *nodes*, which enables quick

access to specific parts of the document – everything is stored in memory as long as a DOM tree exists. Particular nodes correspond to particular data taken from the original document.

Unlike the SAX interface, the document object model has been developed in the W3C consortium. SAX is a public domain software, a result of long discussion in the *XML-dev* mailing list, whereas DOM is a standard in itself, like XML.

2.1. DOM application in Java

In order to be able to use the DOM in a specific programming language, interfaces and classes should be applied and the DOM model itself should be implemented. Since the applied methods are not specified in the DOM specification, it was necessary to develop the language interfaces representing a conceptual structure of the DOM model – both for Java and for other languages. The interfaces enable the user to process documents in the manner described in the DOM specification (*„Java and XML"*, Helion, Brett McLaughlin, 2003.).

Most processors do not independently generate DOM input data. To do this, they use an XML parser, whose task is to generate a DOM tree. Therefore, it is the XML parser rather than the XSLT processor that will have the necessary DOM classes. Since by default Apache Xalan uses the Xerces parser for the DOM model generation and processing, the DOM handling, described below, will take place from the level of that tool.

2.2. How DOM works

In order to familiarise oneself with how the DOM model works, it is best to show the way in which the Apache Xalan processor receives an XML document in a DOM tree-like structure.

The DOM model does not define how a DOM tree is created. The authors of the specification have focused on API structure and interfaces used for manipulating the tree. There is considerable freedom of the DOM parser implementation. Unlike in the SAX XMLReader class, which dynamically loads the implementation, in DOM it is the programmer who has to openly import and create a copy of DOM parser class of a specific manufacturer. Shown below is a way of using DOM in a code fragment. The first thing is to create DOM parser implementation *(listing 1)*.

```java
// Importuje parser DOM
import org.apache.xerces.parsers.DOMParser;
public class MyActionListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
// Tworze egzemplarz implementacji parsera DOM
DOMParser parser = new DOMParser();
try {
parser.parse(path);
                Document doc = parser.getDocument();
                processNode(doc);
} catch (Exception e) {
System.out.println("Błąd w przetwarzaniu: " +
e.getMessage());
}
}
}
```

Listing 1. Creating implementation for the DOM parser DOM.

As is seen in the source code presented above, first the Apache Xerces DOMParser class is imported and its copy is created. The DOM model operation is focused on the output data from processing. The data cannot be used until the entire document has been created and added to the do initial tree structure. The processing output data, which are to be used by DOM interfaces, have the form of a org.w3c.dom.Document. document. The object acts as an "operation manual" for the tree into which the XML data have been input. From the point of view of the element hierarchy, the object is situated one level "above" the main element of an XML document. In other words, each element of the input XML document is directly or indirectly a child element against it.

In order to retain the standard interface in the SAX and DOM parsers, the parse() method, seen in the code, is of the "void" type, like the one used in the SAX model. Owing to it, the application can use a DOM and SAX class parser interchangeably, but it implies the necessity to develop a new method to get the Document object, which is the result of XML processing. In the Apache Xerces parser, the method is called getDocument()(*„Java and XML"*, Helion, Brett McLaughlin, 2003.).

2.3. The DOM tree

In order to show how one can go through the structure of a previously obtained DOM object, it is best to take a initial Document type object and process its every node and all its child nodes. In order to understand the process principle, one should view the basic objects through which access to

XML data will be provided. A Dokument object has been described earlier and presented in Fig. 2 with other basic interfaces of DOM objects (which also shows interfaces used less frequently).

With these interfaces, it is possible to process data within the DOM tree. Particular attention should be paid to the Node interface, as this is the basis for all the others. It is noteworthy that it is possible to develop a method which takes the node, recognises the DOM structure of the node and processes the node in a suitable manner. In this way, the entire DOM tree can be processed with one method. When the node has been processed, available methods can be used to go to the next sibling element, taking attributes (if it is an element) and handling any possible special cases. Subsequently, performing iteration along child nodes, the method is recurrently started on each of the nodes. It is the simplest and clearest method of DOM tree handling.
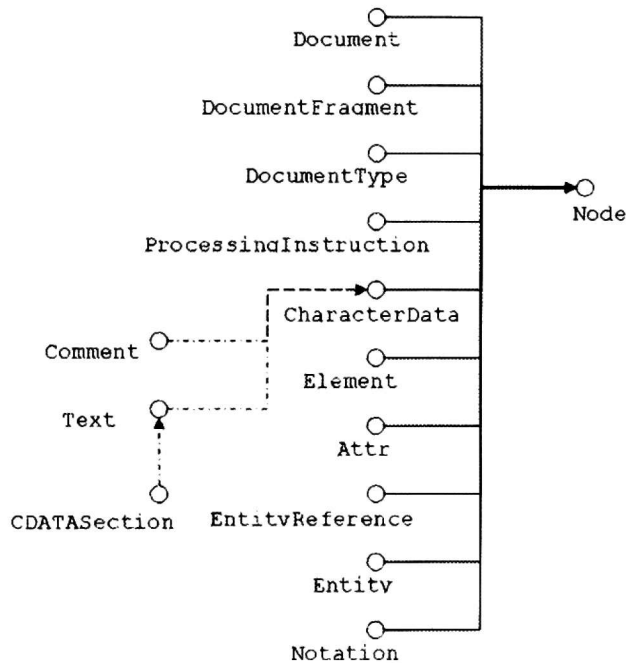

Fig. 2. Presentation of the basic interfaces and classes of DOM Level 2.

Since the Document itself is a Node of the DOM model, it can be transferred unchanged to the processing method. Before a skeleton of such a method has been created, relevant importing instructions have to be added and the taking and processing method for the DOM Node object has to be declared (see *listing 2*):

```
import org.w3c.dom.Document;
import org.w3c.dom.Node;
// Import parsera DOM
import org.apache.xerces.parsers.DOMParser;
 public void processNode(Node node) {
// Rozpoznanie typu węzła
// Przetworzenie węzła
      // Przetworzenie rekurencyjne węzłów potomnych
}
```

Listing 2. Declaration of packages and the DOM tree processing method.

When the method skeleton is in the right place, the method can be run on the initial object Document and you can recursively process the structure until the entire tree has been processed. It is possible thanks to the fact that the Document is a part of the common Node interface.

Finally, a certain drawback of the DOM model should be mentioned. As the entire document is entered in memory, system resources run out quickly, often causing applications to slow down or even preventing them from functioning correctly. Using the DOM model requires engaging the amount of memory in proportion to the size and complexity of the XML document. There is no way in which the demand for memory can be reduced. Moreover, the transformations themselves burden the system on which they are run, which in combination with the memory-related requirements may result in having to use a different way of processing an XML document. If a small document, below one MB, is processed, any problems are unlikely. Larger documents – technical handbooks or entire books – may use up the system resources and affect the application efficiency. Therefore, the choice of the appropriate interface for XML document processing should be made carefully. The choice of the best solution for a specific program project will be decided by the application characteristics.

3. JAXP (Java Api for XML Parsing).

JAXP is a Java interface for XML processing, developed by Sun Microsystems. The aim of the interface is to ensure coherence between the SAX and DOM interfaces. It is not meant as competition for them, nor has it been developed to become their successor. It just offers simplified methods aimed at facilitating the use of Java interfaces in XML document handling. It is compatible with the SAX and DOM specifications and with recommendations regarding namespaces. JAXP does not define how SAX or DOM interfaces should behave, but it provides a standard access layer for all the XML parsers from Java level.(*„Java and XML"* published by Helion, Brett McLaughlin, 2003)

JAXP is expected to evolve as the SAX and DOM interfaces are modified. One can also forecast that it will ultimately take its place among other Sun specifications, as both the Tomcat servlet mechanism and the EIB specification require XML-formatted files for configuration and implementation.

Summary

Two important trends have emerged recently: XML stream processing and XML object mapping. Both seem to be very important for its future. The first is like to replace the SAX model in a longer perspective, while the latter will take the place of DOM. When this will take place depends mainly on how long software manufacturers will support these solutions and to what extent they will be developed and improved.

Abstract

This paper discusses the basic Java interfaces, which are used for XML document handling. Three interfaces have been discussed: SAX – used for sequential document processing, DOM – document object model and JAXP – an interface which imparts coherence to the SAX and DOM techniques. The paper presents their major features, differences and practical applications.

References

1. Arciniegas F., *XML. Kompendium programisty,* Helion, 2002.
2. Eckel B., *Thinking in Java,* Helion, 2001.
3. McLaughlin B., *Java and XML,* Helion, 2003.
4. North S., *XML. dla każdego,* Helion, 2000.
5. Rusty H., *XML. Księga eksperta,* Helion, 2001.
6. Traczyk T., *XML – stan obecny i trendy rozwojowe,* IX Konferencja PLOUG, October 2003.