Research on XML Schema Transformation Algorithm

Bin JI, Jia-ju WU*, Hui-jun LIU, Li-rong MENG and Wan-yi PENG
China Academy of Engineering Physics, Institute of Computer Application, Mianyang, China
*Corresponding author

Keywords: XML Schema, JAVA, Transformation, Integrated support system.

Abstract. An efficient method for transforming XML Schema into programming language is the development basis of XML related software. In this paper, we propose an algorithm design, which is used to transform XML Schema document into Java codes. Firstly, we studied XML Schema elements and nested relation between elements in all cases. Then we studied DOM API and used DOM parser to parse XML Schema document. Thirdly, we studied Deep First Search algorithm and used it to traverse parsed result. During the traversing process, create corresponding Java objects for XML Schema elements. Finally, we got a collection of Java objects, which is organized according to nested relation between elements. Except notation and anyAttribute, the algorithm can transform the other elements contained in XML Schema document into Java objects, which makes sure that the collection of transformational Java objects is an equivalent representation of XML Schema document.

Introduction

Integrated support data of weapon equipment is used in integrated support systems, such as automated information systems for transportation and replenishment, automated information systems for maintenance, etc. [1]. Before used in these systems, integrated support data need to be collected and edited into XML (eXtensible Markup Language) [2] format. For weapon equipment’s integrated support data is highly confidential, so we can’t apply commercial data acquisition and editing software to these systems. Therefore, design and develop a self-controllable and highly automated XML editor is one of key factors to improve weapon equipment automation and digital level. XML Schema is used to design, constrain, and validate XML, and has become the primary choice for designing, constraining, and validating XML [3]. Therefore, it is necessary to design an efficient algorithm to transform XML Schema into programming language, which is the development basis of XML data editor.

Due to the textual nature of XML Schema and the complexity of XML syntax, XML Schema is very complex. Firstly, XML Schema both has rich built-in data types and supports user-defined data types. Secondly, both element declaration and attribute declaration have two types, which are global declaration and local declaration. Thirdly, both simpleType definition and complexType definition have two types, which are name definition and anonymous definition. Fourth, elements can be nested with each other, and the nested hierarchy can be unlimited [4]. Unfortunately, fewer researchers focus on XML Schema transformation. In order to transform XML Schema into Java codes, we studied XML Schema syntax, which mainly includes elements usage and nested relation between elements. In addition, we studied XML Schema document structure. Paper [5] provides an approach to represent XML Schema document with UML, which can help to understand XML Schema document structure quickly. Based on XML Schema syntax study, we propose an algorithm to transform XML Schema into Java codes in this paper. DOM (Document Object Model) [6,7] is used to parse XML Schema document. DFS (Deep First Search) is used to traverse elements contained in XML Schema document. The algorithm can transform XML Schema document into Java objects correctly and effectively. These Java objects are organized according to nested relation between elements, which is an
equivalent representation of XML Schema document. According to the algorithm design, we implement the algorithm implementation, which is shown in paper [8].

**XML Schema Syntax**

There are thirty kinds of elements and twelve kinds of facet elements defined in the latest version of XML Schema [9, 10, 11]. However, according to practical application, two kinds of elements are not taken into consideration in the algorithm design, which are notation and anyAttribute. Similar to elements, some attributes are not taken into consideration too, as shown below.

1) id and anyAttributes of all kinds of elements
2) nillable, block and final of element
3) blockDefault, finalDefault and version of schema

Table 1 shows twenty-eight kinds of elements, elements’ attributes and elements’ sub elements, which are taken into consideration in the algorithm design. Any possible nested relation between elements can be concluded according to Table 1.

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Attribute</th>
<th>Sub Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>null</td>
<td>targetNamespace xmlns</td>
<td>annotation (include import group redefine simpleType attributeGroup complexType element attribute)*</td>
</tr>
<tr>
<td>import</td>
<td>schema</td>
<td>namespace schemaLocation</td>
<td>annotation</td>
</tr>
<tr>
<td>include</td>
<td>schema</td>
<td>schemaLocation</td>
<td>annotation</td>
</tr>
<tr>
<td>redefine</td>
<td>schema</td>
<td>schemaLocation</td>
<td>Annotation (attributeGroup group complexType simpleType)*</td>
</tr>
<tr>
<td>element</td>
<td>schema choice all sequence</td>
<td>substitutionGroup name type default maxOccurs form fixed abstract minOccurs ref</td>
<td>simpleType complexType (unique key keyref)*</td>
</tr>
<tr>
<td>attribute</td>
<td>attributeGroup schema complexType extension restriction</td>
<td>default fixed name ref use type</td>
<td>annotation simpleType</td>
</tr>
<tr>
<td>group</td>
<td>sequence redefine complexType choice extension restriction schema</td>
<td>name minOccurs ref maxOccurs</td>
<td>annotation all choice sequence</td>
</tr>
<tr>
<td>attributeGroup</td>
<td>restriction extension complexType attributeGroup</td>
<td>name ref</td>
<td>annotation (attribute attributeGroup)*</td>
</tr>
<tr>
<td>simpleType</td>
<td>schema union element attribute list redefine restriction</td>
<td>name</td>
<td>annotation list restriction union</td>
</tr>
<tr>
<td>complexType</td>
<td>schema element redefine</td>
<td>name abstract mixed</td>
<td>annotation choice all simpleContent complexContent sequence group (attribute attributeGroup) *</td>
</tr>
<tr>
<td>list</td>
<td>simpleType</td>
<td>itemType</td>
<td>annotation simpleType</td>
</tr>
<tr>
<td>union</td>
<td>simpleType</td>
<td>memberTypes</td>
<td>annotation simpleType*</td>
</tr>
<tr>
<td>simpleContent</td>
<td>complexType</td>
<td></td>
<td>restriction extension annotation</td>
</tr>
<tr>
<td>complexContent</td>
<td>complexType</td>
<td>mixed</td>
<td>restriction extension annotation</td>
</tr>
<tr>
<td>restriction</td>
<td>simpleType</td>
<td>base</td>
<td>annotation length all minInclusive maxInclusive minExclusive choice mxsExclusive maxLength minOccurs sequence totalDigits fractionDigits simpleType pattern enumeration group simpleType (attributeGroup attribute)*</td>
</tr>
<tr>
<td>extension</td>
<td>simpleContent</td>
<td>base</td>
<td>annotation (attribute attributeGroup)*</td>
</tr>
<tr>
<td></td>
<td>complexContent</td>
<td></td>
<td>annotation group all sequence</td>
</tr>
</tbody>
</table>
For the twelve kinds of facet elements, restriction is the only parent element and value is an attribute that must be contained in each facet element. In addition, fixed attribute may be contained in facet elements, and no facet elements have sub elements. The twelve kinds of facet elements are length, maxLength, minLength, maxInclusive, minInclusive, maxExclusive, minExclusive, fractionDigits, totalDigits, enumeration, whiteSpace and pattern.

Algorithm Design

XML Schema document conforms to XML syntax, so both DOM and SAX [12] (Simple API for XML) can be used to parse XML Schema document. DOM is chosen to parse XML Schema document in the algorithm design.

Algorithm Design Idea

The algorithm can be divided into two parts, which are parsing unit and Java object creation unit, as shown below.

Part 1: Parsing unit first implements DOM API with Java language and gets an instance of DOM parser. Then the parser calls the parse (URL url) method to parse the XML Schema document, and the url parameter is the URL path of XML Schema document. The parsing result is an instance of Document. Thirdly, parsing unit gets the first element of parsing result, which is named after n (For a well-formed and non-empty XML Schema document, n is schema actually. For convenience, the element is named after n.). At last, pass n to Java object creation unit.

Part 2: Java object creation unit receives n and gets the localName of n. Then call constructor of the Java class that is corresponding to localName to create a Java object for n.

For a well-formed and non-empty XML Schema document, schema is the root element. Except schema, for each of the other elements contained in XML Schema document, it must have a parent element. Therefore, in Java object creation unit, take schema as root element, and traverse any other elements with DFS, which makes sure that each of the elements contained in XML Schema document can be accessed. The transformation result is a Java object created for schema actually. This Java object contains all Java objects created for the other elements that are contained in the XML Schema document. These objects are organized by Java language according to nested relation between elements.
Algorithm Description

This section describes the executing process of Java object creation unit in detail, as shown below.

Step 1: Receive n and get the localName of n, then call the constructor of the Java class, which is corresponding to localName. If n is schema, then (n, null, null) are passed to the constructor; if n is a facet element, then (n, this) are passed to the constructor; otherwise, (n, this, schema) are passed to the constructor. For parameters, the key word “this” represents the current Java object and the key word “schema” represents the Java object of schema. Execute step 2.

Step 2: Execute assignment statements defined for constructor parameters in constructor. Execute step 3.

Step 3: Execute judgment and assignment statements defined for attributes in constructor. Execute step 4.

Step 4: Get the first sub element of n, which is named after n’. Execute step 5.

Step 5: If n’ is an empty element, execute step 6. If not, then the Java object of n, which is the parent element of n’, is created, execute step 7.

Step 6: If parent element of n’, which is named after n, is schema, the algorithm terminated normally, which represents that a Java object is created for schema. If not, assign the Java object created for n to corresponding variable defined in the Java class, which is defined for the parent element of n. Then, get next sibling element of n, which is named after n’, execute step 5.

Step 7: If n’ is an instance of Element, execute step 1; If not, execute step 8.

Step 8: Get next sibling element of n’, which is named after n”. Execute step 5. Fig. 1 shows the executing process of Java object creation unit.
Conclusion and Discussions

This paper designs an algorithm to transform XML Schema documents into Java codes. The algorithm can transform twenty-eight kinds of elements and the twelve kinds of facet elements, which are defined in XML Schema, into Java codes. Transformational Java objects are organized by Java language according to nested relation between elements. The algorithm is designed according to practical application, so all the elements and attributes applied in integrated support system are taken into consideration to make sure the algorithm can meet integrated support system development requirements. For DOM is taken to parse XML Schema document and DFS is taken to traverse parsing result, so time complexity of the algorithm is O(n) and space complexity of the algorithm is O(n) too. The algorithm can be widely used in the development of various XML related software, which includes XML data editing software, data validating software, data acquisition software etc. All these software can be applied to integrated support systems. For time limited, how to transform notation and anyAttribute into Java codes is not studied and some attributes of elements are ignored. Our future work will focus on these details to prefect the algorithm.
References


