An Algorithm to Transform XML Schema into Java Codes

Bin Ji, Jia-ju Wu*, Yong-hui Xie, Li-rong Meng and Yong-qi Ma
China Academy of Engineering Physics, Institute of Computer Application, Mianyang, China
*Corresponding author

Keywords: XML Schema, JAVA, Element, Class, Object.

Abstract. This paper implements a transformation algorithm, which is used to transform XML Schema into Java codes. Based on the algorithm design, we first defined a Java class, which is used to implement parsing unit of the algorithm. Secondly, we defined a Java class for each of the twenty-eight kinds of elements and twelve kinds of facet elements, and build import or other relation between these classes according to any possible nested relation between elements. These classes are used to create Java objects for elements. Thirdly, in order to increase code reusability or define data type of Java class variables conveniently, we defined a set of Java abstract classes or interfaces. Related Java classes extend or implement these abstract classes or interfaces to realize specific functions. Finally, we performed numerous experiments to validate the algorithm. The algorithm can transform any non-empty and well-formed XML Schema documents into Java objects, and can be applied to any XML related software development.

Introduction

Integrated support systems, such as automated information systems for transportation and replenishment, automated information systems for maintenance, etc. have requirements for XML data editor to edit integrated support data [1]. For these integrated support data is highly confidential, so commercial XML data editing software cannot be applied to these systems. Therefore, design and develop a self-controllable and highly automated XML data editing software is one of key factors to improve weapon equipment integrated support level. XML Schema [2,3,4] is used to design, constrain and validate XML [5] and has become the primary choice for constraining and validating XML in integrated support systems [6]. Therefore, an efficient algorithm, which is used to transform XML Schema into programming language, is the development basis of XML data editing software and other XML related software [7].

In paper [8], we design a transformation algorithm. The algorithm is divided into two parts, which are parsing unit and Java object creation unit. Parsing unit first implements DOM (Document Object Model) API [9] and parses XML Schema document with DOM parser. Then parsing unit gets the first element of parsing result and passes it to Java object creation unit. Java object creation unit first receives the first element, then takes this element as root element and traverses the other elements with DFS algorithm. During the traversing process, create corresponding Java objects for elements. Based on the transformation algorithm design, we implement the algorithm with Java language in this paper. In addition, we performed numerous experiments to validate this algorithm to make sure that the algorithm is correct and effective. Based on the algorithm, we developed a prototype of XML data editing software. The prototype has been applied to an IETM (Interactive Electronic Technical Manual) making platform.

Algorithm Implementation

According to the algorithm design, algorithm implementation is divided into two parts: parsing unit implementation and Java object creation unit implementation. A Java class named after Create_XSD_Objects is defined to realize parsing unit. Next, we will describe Java object creation unit implementation in detail. We first define a set of classes and interfaces in order to increase code
reusability or define data type of Java class variables conveniently. Then for each of the twenty-eight kinds of elements, we define a Java class for it. At last, for all the twelve kinds of facet elements, we define a common Java class for them. Detailed procedure is shown below.

Define Special Java Classes or Interfaces

WXSAnnotated Abstract Class
Except annotation, documentation, and appinfo, annotation can be a sub element of any element. In order to increase code reusability, we define an abstract class, which is named after WXSAnnotated. Except the Java classes defined for above three elements, all the Java classes defined for elements and facet elements extend WXSAnnotated.

WXSExplicitGroup Abstract Class
Sequence and choice can nest with each other. The parent elements, attributes and sub elements of the two elements are the same. In order to increase code reusability, we define an abstract Java class, which is named after WXSExplicit. The Java classes defined for above two elements extend it and call method defined in it to transform the two elements into Java objects.

WXSKeybase Abstract Class
The parent element and sub elements of key, keyref, unique are the same. The difference is that refer is an attribute of keyref. In order to increase code reusability, we define an abstract Java class, which is named after WXSKeybase. The Java classes defined for above three elements extend it and call method defined in it to transform the three elements into Java objects.

WXSThing Interface
In order to define data type of Java class variables conveniently, we define an empty interface, which is named after WXSThing. All the Java classes defined for the algorithm implement it. Four usage cases of the interface are shown below.

1) Define a variable named after attributes in the Java classes defined for group, complexType, complexType, and restriction, and the data type of it is List<WXSThing>. The variable is defined for attribute and attributeGroup, which are possible sub elements of above four elements.
2) Define a variable named after identityConstraints in the Java class defined for element, and the data type of it is List<WXSThing>. The variable is defined for key, keyref and unique, which are possible sub elements of element.
3) Define a variable named after redefinables in the Java class defined for redefine, and the data type of it is List<WXSThing>. The variable is defined for simpleType, complexType, group, and attributeGroup, which are possible sub elements of redefine.
4) Define a variable named after modelGroup in WXSExplicitGroup, and the data type of it is List<WXSThing>. The variable is defined for element, sequence, choice, group, and any, which are possible sub elements of sequence and choice.

Define Java Class for Twenty-Eight Elements
Define a Java class for each of the twenty-eight elements. A combination of “WXS” and capitalized element name is taken as class name, such as “WXSSchema”. Establish import relation or other relation between Java classes according to nested relation between elements. Detailed procedure is shown below.

Define Variables for Element’s Attributes in Java Class
For each of element’s attributes, define a variable in the element’s Java class. Generally, variable name is named after attribute name. Data type of variable is String and initial value is null. Some special cases are shown below.
1) `memberTypes` is an attribute of union. The value of `memberTypes` can be a combination of several `simpleType` names, which are separated by whitespace. The data type of `memberTypes` variable defined in WXSUnion is `String[]` and the initial value is null.

2) The `xmlns` attribute of schema has two usage cases: One is to represent default namespace, which is named after `xmlns`. The two is to represent namespace with prefix. In this case, attribute name starts with “xmlns:” which is followed with namespace prefix. Default namespace appears at most one times and namespace with prefix can appear 0 or several times in schema. For default namespace, define a variable named after `xmlns`, and take `String` as data type. For namespace with prefix, define a variable named after `nameSpace`, and take `HashMap<String, String>` as data type. `<String, String>` corresponds to `<namespace, namespace prefix>`. Initial value of the two variable is null.

3) The other special variables defined for attributes are shown in Table 1.

Table 1. The other special variables defined for attributes.

<table>
<thead>
<tr>
<th>Java Class</th>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WXSAll</td>
<td>minOccurs</td>
<td><code>int(1)</code></td>
</tr>
<tr>
<td>WXSC_choice</td>
<td>maxOccurs</td>
<td></td>
</tr>
<tr>
<td>WXSAAny</td>
<td>namespace</td>
<td><code>String(#any)</code></td>
</tr>
<tr>
<td>WXSSequence</td>
<td>processContents</td>
<td><code>String(strict)</code></td>
</tr>
<tr>
<td>WXSGroup</td>
<td>mixed</td>
<td><code>Boolean(null)</code></td>
</tr>
<tr>
<td>WXSCComplexContent</td>
<td>abstract</td>
<td><code>Boolean(null)</code></td>
</tr>
</tbody>
</table>

Attention: in data type column, the value in parenthesis is initial value.

Define Variables for Element’s Sub Elements in Java Class

As described above, some variables are defined for some sub elements in corresponding Java classes. Except these sub elements, for the rest, definition rules are shown below.

1) If the sub element can appear 0 or 1 time in its parent element, define a variable in the Java class defined for parent element. The variable name is named after sub element name. Variable data type is the Java class defined for sub element.

2) For the rest, define variables according to Table 2.

3) Initial value of all variables defined for sub elements is null.

Table 2. Some Variables defined for sub elements.

<table>
<thead>
<tr>
<th>Java Class</th>
<th>Sub Element</th>
<th>Variable</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WXSAll</td>
<td>element</td>
<td>elements</td>
<td><code>ArrayList&lt;WXSElement&gt;</code></td>
</tr>
<tr>
<td>WXSAnnotation</td>
<td>documentation</td>
<td>documentations</td>
<td><code>List&lt;WXSDocumentation&gt;</code></td>
</tr>
<tr>
<td>WXSKey</td>
<td>field</td>
<td>fields</td>
<td><code>List&lt;WXSField&gt;</code></td>
</tr>
<tr>
<td>WXSRestriction</td>
<td>facet</td>
<td>facets</td>
<td><code>List&lt;WXSFacet&gt;</code></td>
</tr>
<tr>
<td>WXSSchema</td>
<td>include</td>
<td>includes</td>
<td><code>ArrayList&lt;WXSInclude&gt;</code></td>
</tr>
<tr>
<td></td>
<td>import</td>
<td>imports</td>
<td><code>ArrayList&lt;WXSImport&gt;</code></td>
</tr>
<tr>
<td></td>
<td>redefine</td>
<td>redefine</td>
<td><code>ArrayList&lt;WXSRedefine&gt;</code></td>
</tr>
<tr>
<td></td>
<td>simpleType</td>
<td>simpleTypes</td>
<td><code>LinkedHashSet&lt;String, WXSSimpleType&gt;</code></td>
</tr>
<tr>
<td></td>
<td>complexType</td>
<td>complexTypes</td>
<td><code>LinkedHashSet&lt;String, WXSCComplexType&gt;</code></td>
</tr>
<tr>
<td></td>
<td>group</td>
<td>groups</td>
<td><code>LinkedHashSet&lt;String, WXSGroup&gt;</code></td>
</tr>
<tr>
<td></td>
<td>attributeGroup</td>
<td>attributeGroups</td>
<td><code>LinkedHashSet&lt;String, WXSAAttributeGroup&gt;</code></td>
</tr>
<tr>
<td></td>
<td>element</td>
<td>elements</td>
<td><code>LinkedHashSet&lt;String, WXSElement&gt;</code></td>
</tr>
<tr>
<td></td>
<td>attribute</td>
<td>attributes</td>
<td><code>LinkedHashSet&lt;String, WXSAAttribute&gt;</code></td>
</tr>
</tbody>
</table>

Define Special Variables in the Java Class

Define three variables in each of the Java classes defined for elements. The first variable is element and the data type is Element. The second variable is parent and the data type is WXSThing. The third variable is schema and the data type is WXSSchema. Initial value of the three variables is null.
Define Constructor in Java Class
Each of the Java classes defined for elements has a constructor. Constructor has three parameters, which are (Element element, WXSThing parent, WXSSchema schema). In constructor, define judgment and assignment statements for each of element’s attributes and sub elements.

Define Java Class for the Twelve Kinds of Facet Elements
Restriction is the only parent element of all facet elements and all facet elements have the same possible attributes. In order to improve the reusability of code, define a common Java class, which is named after WXSFacet. Detailed procedure is shown below.

1) Define a variable named after facet for facet element name, and a variable named after value for facet element’s value attribute, and a variable named after fixed for facet element’s fixed attribute. Data type of facet and value is String and data type of fixed is Boolean. In addition, define another two variables in WXSFacet. The first variable is named after element and the data type is Element. The second variable is named after parent and the data type is WXSThing. Initial value of the five variables is null.

2) Define constructor for WXSFacet. Constructor has two parameters, which are (Element element, WXSThing parent). In constructor, define judgement and assignment statements for each of facet element’s attributes and define assignment statements for facet element name.

Transformation Example
In this section, we take descriptSchema.xsd, which is defined in ATA/ASD/AIA S1000D Issue 4.2 [10] as an example to display the algorithm. In order to show the information of transformational Java objects, we print Java objects information out. For the outputting information is too much to display completely, we cut out a segment, as shown in Fig.1. By comparing the outputting information with descriptSchema.xsd, we found that the algorithm can correctly transform each of the elements contained in descriptSchema.xsd into Java objects and these objects are correctly organized according to nested relation or other relation between elements.

Figure 1. Part of transformational Java objects information.
Results
This paper implements an algorithm to transform XML Schema document into Java objects based on the algorithm design. The algorithm comprises thirty-four Java classes or interfaces, twenty-eight of which are defined for twenty-eight kinds of elements, WXSFacet is defined for twelve kinds of facet elements, Create_XSD_Objects is defined to implement parsing unit and is the entrance of the algorithm. For the rest, they are defined to increase the reusability of code or define data type of variables conveniently. We chose the nineteen XML Schema documents defined in ATA/AIA/ASD S1000D Issue4.2 to validate the algorithm. These documents are defined for Interactive Electronic Technical Manual [11] data modules, and neither anyAttribute nor notation is contained in these documents. We compared transformational Java objects information with original documents. Comparison results proved the correctness and effectiveness of the algorithm. Time complexity of the algorithm is O(n) and space complexity of the algorithm is O(n) too.

Conclusion and Discussions
This paper implements an algorithm to transform XML Schema documents into Java codes based on the algorithm design. The algorithm can transform XML Schema document into Java objects correctly and effectively. Correctness and effectiveness of the algorithm gets validated comprehensively by performing a large number of experiments. The algorithm is designed and implemented according to practical application, so all the elements and attributes applied in integrated support system are taken into account to make sure the algorithm can meet integrated support system development requirements. The algorithm can be widely used in the various XML related software development. Based on the algorithm, a prototype of XML data editing software has been developed and is already applied to an Interactive Electronic Technical Manual making platform.

References