XML Editor Factory

Béla Vander Voort and Kenny Hunt
Computer Science Department
University of Wisconsin La Crosse
La Crosse, WI 54601
vandervo.bela@students.uwlax.edu

An automatically generated XML Editor

XML and XSD provide a hardware and software independent means of exchanging data. Converting data to XML allows it to be more efficiently interpreted by other systems. This paper discusses an application which can generate an editor from a given XSD document. XML documents which conform to the given XSD may then be created with the editor. The details of what this application supports and how this was accomplished are discussed. Also included are current limitations and future work that can be done.

1 Introduction

Computer systems and databases often contain data in a variety of incompatible formats. Exchanging data between such systems is an important and difficult challenge for developers. Converting the data to an Extensible Markup Language(XML) document is one way to allow the data to be read by a wide variety of applications. XML is written in plain-text with tags to describe the data. Each set of tags defines an element and its attributes, while the element's data is between the two tags. Figure 1 shows a sample XML document which contains the data for a person.

```
<person gender="male">
   <firstname>John</firstname>
   <lastname>Doe</lastname>
</person>
```

Figure 1: sample xml document

Problems may arise if another XML document describes a person with the first and last name tags in the opposite order. To avoid this and allow data to be efficiently interpreted XML documents need some defined structure. This structure can be defined using XML Schema Definition (XSD) documents. XSD documents are written in XML therefore both types of documents can be manipulated using the same tools.

XML and XSD provide a platform independent method of exchanging data, however due to the complexity an XSD may take on it can become a very challenging task to write an XML file that conforms to it. Software tools can be used to simplify this task. An application can be written to create XML files for a specific XSD, however this application must be modified each time the XSD is altered. Another solution to this problem is an application that uses an XSD document as input and builds an editor based on this document. A user is then able to create XML files that conform to the original XSD. This paper discusses just such an application including the features it supports, how they were implemented, limitations of the application, and future work to be done.

2 Project Description

Our application, XEF, is made up of three different windows. The main window contains components that represent the XSD document's elements. Attributes of these elements are displayed in a separate window to keep the main window less cluttered. The third window is an informational window. This window is used to display information about the current element.

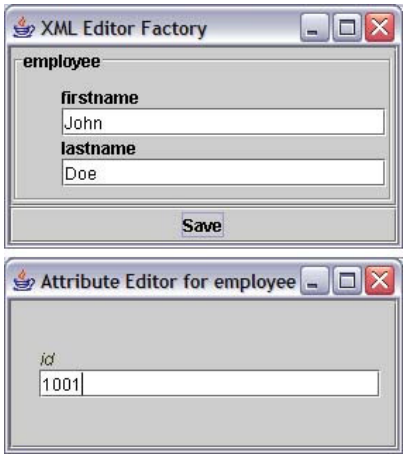
Figure 2 gives an example of how XEF functions. Initially XEF is given a XSD document and the name of an element to use as a starting point. The XSD document given to XEF is displayed in Figure 2a. This document defines the structure of an XML

document for an employee. An employee element is made up of two elements, firstname and lastname, and one attribute, id. Figure 2b displays the editor that XEF would build from this XSD document. The resulting XML document output by the editor is displayed in figure 2c.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="employee">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="firstname" type="xs:string"/>
         <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
  </xs:complexType>
</xs:element>
```

(a)



(b)

```
<?xml version="1.0" encoding="UTF-8"?>
<employee id="1001"><firstname>John</firstname><lastname>Doe</lastname></employee>
```

(c)

Figure 2: A sample XSD document defining an Employee is given in (a). Screenshots showing the editor that results from inputting this document into XEF are given in (b). The XML file that would be output after inputting the values displayed is given in (c).

3 Technical Details

This section describes the implementation details of XEF. The application is written in Java and decomposed into two central classes. The main purpose of the XMLEditorFactory class is to use the input XSD document to construct a base for the XML document being created and to generate the editor. The editor itself is primarily made up of subclasses of the XSDComponentEditor class. A UML class diagram of XEF's classes is displayed in figure 3.

XEF also makes use of the org.eclispe.xsd and org.w3c.dom packages. The org.eclise.xsd package provides a mapping from XSD specifications to java classes. An XSD document and all of its components can be mapped to subclasses of the base XSDComponent class. The org.w3c.dom package provides classes for the XML Document Object Model(DOM). The DOM represents a tree view of the XML document. This tree is made up of nodes which represent the different elements and attributes that appear in the XML document.

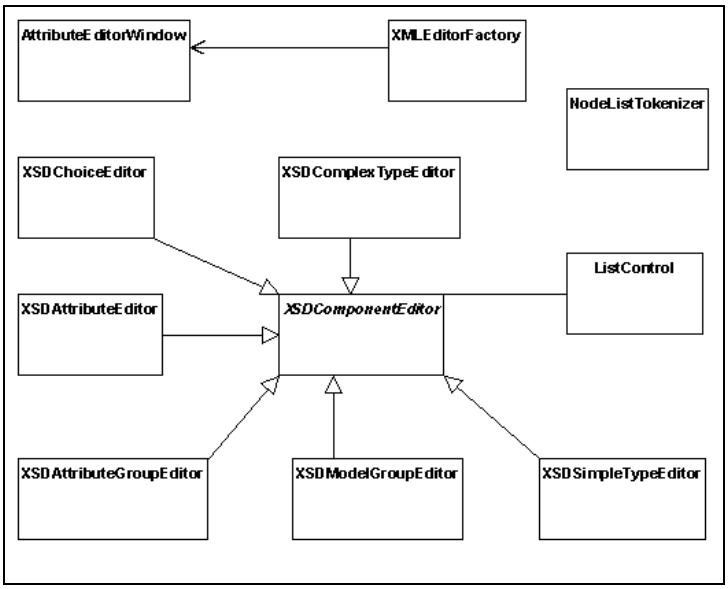


Figure 3: UML Class Diagram for XEF

Initially XEF uses a method from the XMLEditorFactory class to convert the XSD document into an XSDSchema object. This is an XSDComponent which represents the entire XSD document. An element name is also needed when XEF loads. The XSDComponent corresponding to this element is then passed to a recursive method which constructs a DOM from a given XSDComponent. XEF also uses a recursive method to generate different components of the editor from a given node of a DOM and its corresponding XSDComponent. Each of these component corresponds to a node or nodes in the DOM and as the user enters information and manipulates the GUI the DOM is updated. When the user wishes to output an XML file the nodes of the DOM tree are traversed and the appropriate elements and values are written out to file.

An XSD document will define certain types of content that may appear in an XML document. The following sub-sections describe the types of elements that may appear in an XSD specification and the way each is handled in XEF.

3.1 Simple Types

A simple element is an XML element that can contain only text, it may not contain other elements or attributes. Simple elements are usually displayed by a textbox with the name of the element above it. Simple elements can be given a default value in the XSD document, XEF will set any default values initially which the user may then modify if desired. Their values may also be declared as fixed, meaning they may not be changed. XEF enforces this by simply making the textbox read only.

An XSD document may give a simple element a list of values it is constrained to, known as enumerated values. XEF handles enumerated values by displaying a dropdown menu

enabling the user to select the desired value. A screenshot illustrating an editor containing a simple element with enumerated values is shown in figure 3.1 below. In this case "position" has the possible values of "entry level", "shift lead" or "manager".

Figure 3.1: A screenshot of XEF that contains a dropdown menu with the enumerated values possible for the simple element position.

Attributes are also a simple type. Much like simple elements, attributes only contain text which can be declared with a fixed or default value. They are also represented with a text box below the name of the attribute.

3.2 Complex Elements

Complex elements are elements that contain more than just a text value. They may be empty, contain only other elements, contain only text, or contain both elements and text. Any complex element may contain attributes as well. XEF represents them with a border around their contents and with the name of the element in the upper left. Complex elements may have indicators which control how they are to appear in the XML document. The following sub-sections describe how XEF handles these indicators.

3.2.1 Order Indicators

Order indicators define how the child elements of a complex element should occur. The most basic of these is the sequence indicator. This indicates that the child elements must appear in the same order in the XML document as they appear in the XSD document.

A complex type with an all indicator specifies that the child elements of the complex element can appear in any order. This poses a problem because XEF cannot rely on the order of the elements in the XSD document. Initially they are generated in that order but a user may alter this order by using a right click menu to move them up or down. This modifies the order of the corresponding nodes in the DOM. When XEF needs to regenerate this part of the editor it must then follow the order of the DOM and match those nodes up with their correct XSDComponent. Figure 3.2.1a illustrates this feature. The contents of employee may occur in any order and the editor allows the user to rearrange them. In this case the user has the option of moving "lastname" up or down.

Figure 3.2.1a: A screenshot of XEF in which a complex element, “employee”, has an all indicator and so the user may rearrange its content’s by right clicking on them to bring up a menu.

The final order indicator, choice, indicates that the complex element contains only one its child elements. Choice indicators are handled by allowing the user to select which element to use with a right click menu. The editor for the selected element is then generated and added. Tracking which choice a user made when the editor is regenerated poses some problems. XSDComponents cannot be altered so we use the names of nodes in the DOM tree to track the users choice. Before the choice is made a generic “Choice” node is created which initially has no nodes appended. When a choice is made a “selection” node is appended to the “Choice” node with a name that indicates which choice the user has made, the actual node corresponding to this choice is then appended to the selection node. When the editor generation method encounters this choice node within the DOM it can use the child node to determine which choice was made, if any.

Figure 3.2.1b illustrates this feature. In this example employee is a complex element with a choice indicator. The possible elements that may appear within employee are “entrylevel”, “shift leader”, or “manager”. On the left the user right clicks on the choice to bring up a menu which allows them to make their choice. On the right the resulting editor is displayed within choice. The user has also right clicked on again to bring up the option of removing their choice so that they may alter it.

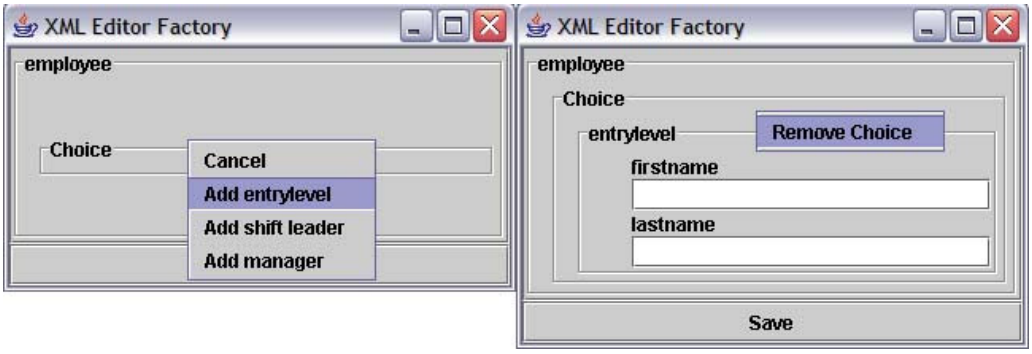


Figure 3.2.1b: Screenshots of XEF illustrating how choice indicators are handled. The user is allowed to make a choice of which element occurs with a right click menu on the left. On the right the choice has been made and a right click menu now allows them to remove the choice.

3.2.2 Occurrence Indicators

Occurrence indicators define how many times an element may occur. XSD uses the keywords minOccurs and maxOccurs to define the limits to how often an element may occur, which may be from zero to unlimited. When XEF encounters an element that may occur more than once it will add a ListControl component to that element's editor. This ListControl contains buttons allowing the user to add and delete another instance of the element as well to navigate through the existing instances of it. An editor is only displayed for one instance of the element. When the user navigates through elements using the ListControl the GUI needs to be regenerated. This is due to that fact that different instances of elements in the DOM may not contain the same elements. Figure 3.2.2 illustrates the ListControl feature. In this case the "employee" element currently occurs three times and the second occurrence is displayed. The buttons in ListControl allow the user to navigate to the first instance, move to the previous instance, add a new instance, delete the current instance, move to the next instance or navigate to the last instance.

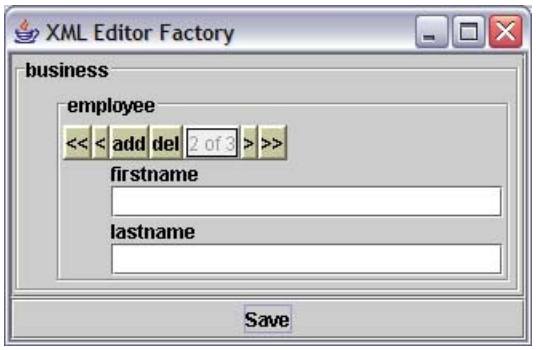
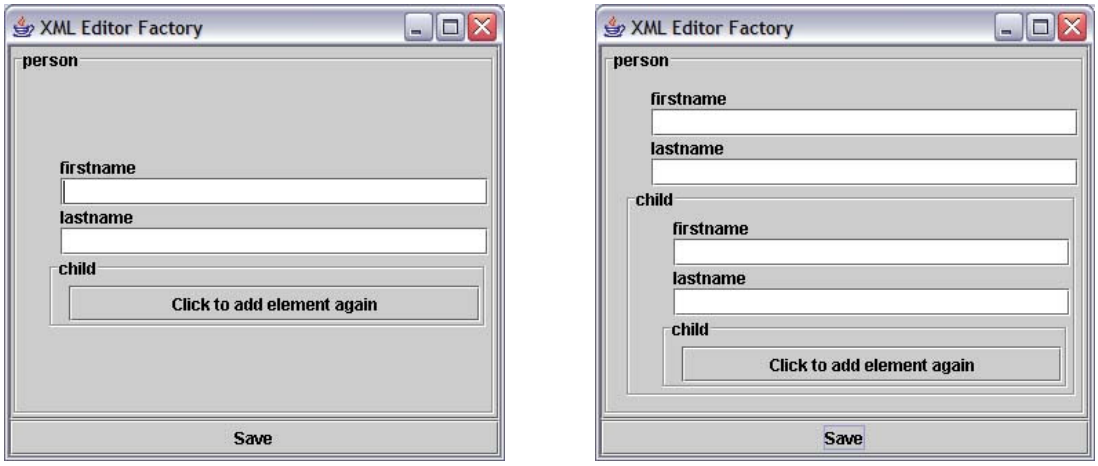


Figure 3.2.2: Screenshot of XEF with a ListControl. Employee may occur more than once and so the ListControl is displayed. The user currently has three instances of the element and the second instance is displayed in the editor.

3.2.3 Elements containing themselves

It is possible that a complex element may contain itself or that another element underneath it may contain a reference to it. There is no special indicator in the XSD document to define this and so the node generation method needs to check for this to avoid constructing an infinite DOM tree. When XEF encounters an element that occurs inside of itself it constructs a placeholder node to add to the DOM tree instead. When the editor generation method encounters this placeholder node it generates a button which allows the user to add another occurrence of this element. Each new occurrence will of course contain another button and the user may continue to add the element as many times as needed. Figure 3.2.3 illustrates this feature. In this case the person element may contain a child element. Child is a reference to the person element and so a child may have a child and so on. Instead of adding another editor for child XEF instead adds a button allowing the user to do this, shown in (a). In (b) the user has clicked the button to add a child to the initial person, this child may also contain a child and so another button is added to allow the user to do this.



(a) (b)

Figure 3.2.4: Screenshots of XEF with an element that contains itself. (a) illustrates an editor for person which contains a child. A child is simply another person element and so a button is displayed instead of another editor. In (b) the user has added the child to person.

4 Results

Illustrated in figure 4.1 is an editor that was generated for an XSD for landXML, an extremely complex XSD. Figure 4.2 is the resulting XML file that was output after entering data into the editor. It has been formatted so that it is readable.

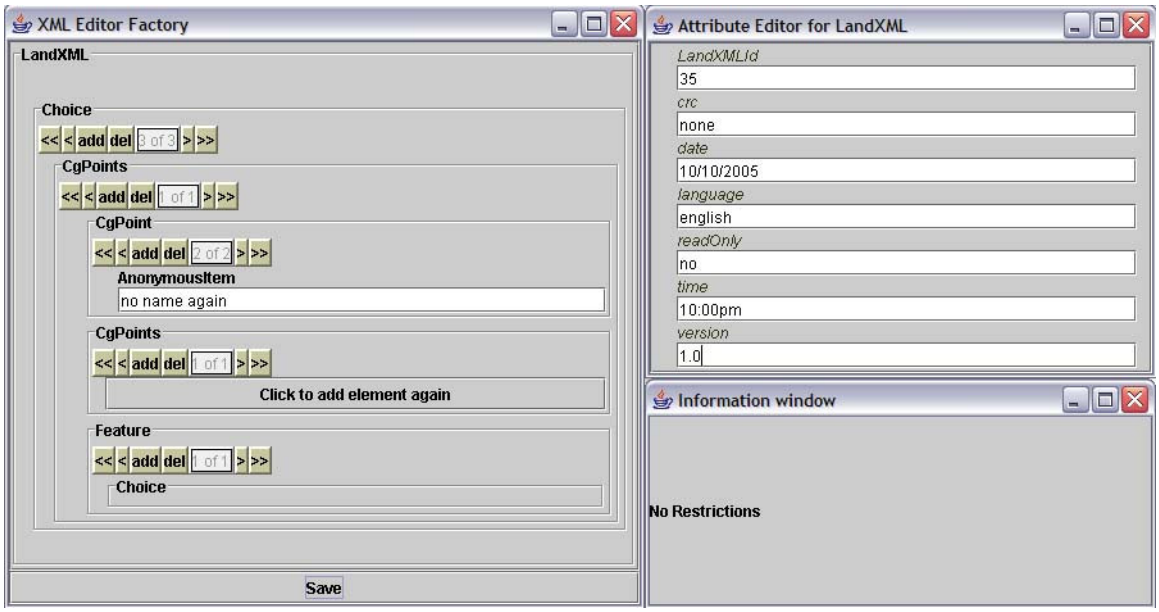


Figure 4.1: XEF screenshots of the editor generated for the landXML XSD.

```
<?xml version="1.0" encoding="UTF-8"?>
<LandXML LandXMLId="35" crc="none" date="10/10/2005" language="english" readOnly="no"
time="10:00pm" version="1.0">
    <Monuments desc="some monuments" name="a name" state="WI">
        <Monument beacon="this beacon" beaconProtection="none" category="high" condition="bad"
        desc="some monument" name="the monument" oID="12" pntRef="12" state="WI"
        type="state"/>
        <Monument beacon="another beacon" beaconProtection="yes" category="low"
        condition="very good" desc="a good monument" name="the tall monument" oID="12"
        pntRef="12" state="WI" type="stone"/>
        <Feature code="12" source="the source"/>
    </Monuments>
    <Units>
        <Metric angularUnit="radians" areaUnit="meters" diameterUnit="meters"
        directionUnit="radians" flowUnit="flow units" heightUnit="meters" linearUnit="inches"
        pressureUnit="pressure" temperatureUnit="degrees" velocityUnit="kilometers"
        volumeUnit=""/>
    </Units>
    <CgPoints DTMAttribute="The attribute" code="12" desc="Some cgpoints" name="The cgpoints"
    state="WI" zoneNumber="15">
        <CgPoint oID="12" pntSurv="35" surveyOrder="first" zoneNumber="12">
            <AnonymousItem>No name</AnonymousItem>
        </CgPoint>
        <CgPoint oID="68" pntSurv="13" surveyOrder="second" zoneNumber="87">
            <AnonymousItem>no name again</AnonymousItem>
        </CgPoint>
        <Feature code="12" source="none"/>
    </CgPoints>
</LandXML>
```

Figure 4.2: XML document generated by XEF for the landXML XSD. Formatted for readability.

5 Limitations

XEF still contains a number of limitations. Complex elements currently do not support mixed content, meaning text is allowed to appear between the child elements of a complex element. XSD also allows complex elements to contain <any> element. This means that any element that appears in any XSD may appear in place of the <any>. Similarly, a complex element can be declared with <anyAttribute> meaning any attribute from another XSD may appear in it. XEF currently does not have a means for the user to include <any> elements or <anyAttributes>.

XSD documents can also be written to include two different sets of element names that define the same data. For example, if you would like to let the XML document contain either Spanish or English element names. XEF only allows the default element names and does not allow the user to substitute the alternate names.

Simple elements and attributes are defined with types. Some common XSD types include xs:string, xs:decimal, xs:integer, xs:boolean, and xs:date. Currently the user may input any type of string for their values, no type checking is performed. Simple elements and attributes may also have different restrictions on their values. Currently enumerated values are implemented using a dropdown menu to allow the user to select the value. Other unsupported restrictions may include a min or a max for an integer value, and a pattern value defining a series of letters or numbers that can be used and what pattern they must follow.

6 Conclusion and Future work

The biggest challenge in our approach was correctly manipulating the underlying DOM and getting the XSDComponents matched to the correct nodes of the DOM. The nodes are the only place new information can be stored, and the information they can hold is limited. These nodes needed to be matched to the correct XSDComponents to correctly generate and update the editor. Thus far this limitation didn't prove too great but an alternative solution could be to develop a middle layer between the editors and the DOM. This layer could be used to store all the relevant information from the XSDComponents and keep them matched to the correct node in the DOM.

Further work could remove the limitations of XEF discussed earlier and get it to fully support all the features of XSD. Other enhancements could include a syntax checker to ensure that the loaded XSD file has no errors as well as outputting XML files with some formatting. XEF could also be extended to allow a user to load a current XML file into the generated editor. Other features could be added to allow the user to define the appearance of the resulting editor.