XCG: A ONE-TO-ONE CLASS GENERATOR FOR XML LANGUAGES

Henrike Schuhart
Institut für Informationssysteme, Universität zu Lübeck
Ratzeburger Allee 160, D- 23538 Lübeck
schuhart@ifis.uni-luebeck.de

Martin Lipphardt
Institut für Informationssysteme, Universität zu Lübeck
Ratzeburger Allee 160, D-23538 Lübeck
Lipphardt@gmx.de

Volker Linnemann
Institut für Informationssysteme, Universität zu Lübeck
Ratzeburger Allee 160, D-23538 Lübeck
linnemann@ifis.uni-luebeck.de

ABSTRACT

High-level XML integration into object oriented programming languages becomes more and more important for the development of web applications. In this paper we present a new **X**ML **C**lass **G**enerator for Java (XCG), which transforms XML language descriptions like DTDs into an equivalent framework of Java classes and interfaces preserving the structure of XML elements. Moreover we present measurements demonstrating the efficiency of the generated class framework in context of XPath queries and related approaches like Castor and XMLBeans.

KEYWORDS

XML and other Extensible Languages, Object Orientation.

1. INTRODUCTION

The World Wide Web (WWW) is the biggest infrastructure for information publishing and exchange. In this context XML as standard format for documents becomes more and more important. In most cases XML documents are generated automatically from databases. To exploit the content of any XML structured file, it is important to find an efficient representation. One common representation is the Document Object Model (DOM) (W3C,1998), which has the advantage that every XML document can be gathered. On the contrary, accessing single elements is very inefficient, because knowledge about the XML document's specific structure is not supported by the DOM. Thus an XPath query selecting child elements with a certain name can only be evaluated by testing all siblings, which results in linear access time. Since most XML documents are structured according to a given XML language description like XML Schema (W3C,2001) or DTD (W3C,2004), it is more efficient to use so called class generators to produce classes which are capable to represent XML documents of a specific XML language much more efficiently. In this paper we will introduce a new **X**ML **C**lass **G**enerator for Java (XCG), which offers efficient access to sub elements in constant time, provides a simple class structure and moreover guarantees universality. In this context universality means that every correct DTD or XML Schema can be mapped onto (Java) classes.

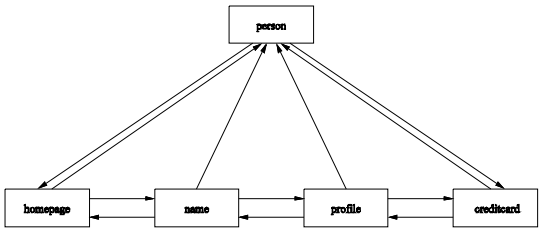
This paper is organized as follows: In section 2 we present related work. In section 3 we introduce foundations and design of our new class generator. In section 4 we present experimental results comparing our class generator with related approaches including JAXB (Ort and Mehta,2004), Castor (Exolab,2004),

XMLBeans (Apache,2004) and the DOM in context of the XMark Benchmark (CWI,2003). Finally in section 5 we give an outlook on future work and finish with a conclusion in section 6.

2. RELATED WORK

DOM and SAX. The W3C's Document Object Model (DOM) (W3C,1998) is a collection of language independent application programming interfaces. The DOM offers standardized interfaces to represent structure and content of any XML document. An implementation of the DOM allows the access and the manipulation of XML documents, e. g. Apache's Xerces implementation (Apache,2003). The idea is to represent XML documents as trees, which are themselves a hierarchy of *Node* objects. The *Node* is the super interface all specialized interfaces or classes inherit from. A *Node* has got references to its parent, its following and preceding sibling as well as to its first and last child as shown in figure 1. In general a *Node*'s children are registered in a list. The simplicity and universality of the DOM are its advantage as well as its disadvantage. Due to simplicity accessing certain child elements is very inefficient.
The Simple API for XML (SAX) (XML) was originally a Java only API and does not offer a representation for a whole XML document. Instead SAX parses a document and causes events, by this technique it is possible to access much larger XML documents as with DOM.

Figure 1.Representation of a person with the W3Cs DOM



Class Generators. Class generators map elements of an XML language description like XML Schema (W3C,2001) or DTD (W3C,2004) on to classes of an object oriented programming language, mostly Java. Attributes and children of an XML element become member variables of the generated classes. This mapping is called *marshalling*, the process from classes to text-based XML is called *unmarshalling*. Additionally methods can be added to the generated classes to offer efficient access. The Java API for XML Binding (JAXB) (Ort and Mehta,2004) is a tool for processing XML data as Java objects. JAXB provides a framework for marshalling, unmarshalling and validation. Recently, a new version was published which only supports XML Schema and no longer DTD's.
Castor (Exolab,2004) is an open source project generating Java classes from an XML Schema. Castor includes a mapping framework between XML documents and Java classes. Additionally, schema free mapping between XML documents and Java objects is possible. XMLBeans (Apache,2004) is another tool for XML integration into the Java programming language. XML Schemas are used to generate Java classes and interfaces.
In contrast to our approach JAXB as well as Castor do not allow the occurrence of more than one same typed sub element. The valid element definition <! ELEMENT A B,B> for instance, which declares the content of A to be a sequence of two elements B, is rejected by JAXB and Castor. Choice types are not reflected by the generated class structure of Castor and JAXB, again this is in contrast to our approach.

3. THE CLASS GENERATOR

The new class generator XCG introduced in this paper is supposed to generate Java classes from XML language descriptions like XML Schema or DTD. Moreover, desirable characteristics include accessing sub elements more efficiently, producing a simple class and interface structure and offering universality in

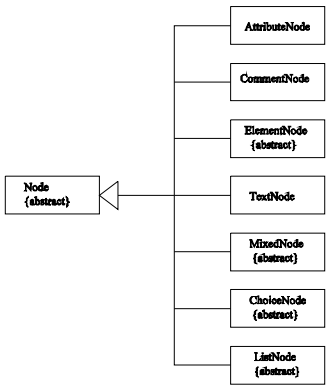
respect to correct XML language descriptions. In addition the classes support XPath specific accesses like node tests. Node tests provide all (sub) elements of a given name and or type. Explicit support of node tests is achieved by generating the method **getChildNodesOfType** for each class**.** The information about the elements structure is implemented in this method. Thus the requested elements can be returned instantaneously. The method expects a node test as parameter and returns a node list. Result types of any element request should be of uniform type, so that query results can be processed consistently.

Direct access of sub elements is supported implicitly by generated classes, because sub elements become members of the new classes. Get and set methods are named according to the index of child elements or attributes. To guarantee a uniform query result type, the DOM's idea that everything is a node is adopted.

Basic Classes

Generated classes implement and or inherit from common basic classes. These classes and interfaces provide basic functionalities in context of XML. Moreover, some of them are used to represent specific XML structures like lists or choices. Beside the *Node* class there are *AttributeNode, CommentNode, ElementNode, TextNode, MixedNode, ChoiceNode* and *ListNode.* The inheritance hierarchy is shown in figure 2. The return type of every element query is defined to be always *Node*, which is the uniform type we demanded.

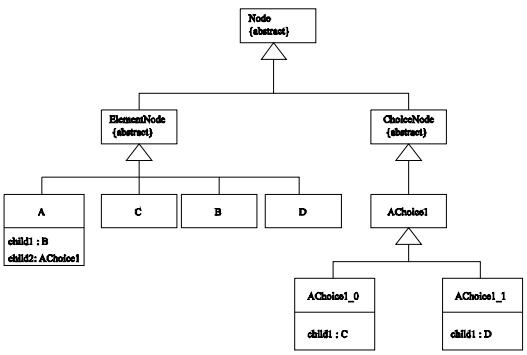
Figure 2.Inheritance hierarchy of the basic classes



Structure Representation

In order to provide universality equivalent to the DOM, it is essential to map any nested structures which could be formulated with DTDs and XML Schema to the generated Java classes.

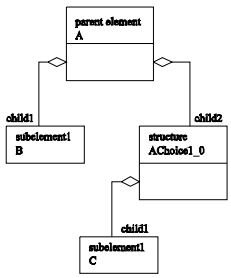
Figure 3.Class diagram of basic and generated classes for the example element description <! ELEMENT A B, (C|D)>



Sequences of sub elements and attributes are mapped directly to indexed member variables as mentioned before. Consequently we will concentrate on lists (Kleene-Star operator), union (Choices) and mixed content.

The basic idea is to offer abstract classes for each operator, i.e. *ListNode, ChoiceNode* and *MixedNode.* Contrary to the DOM, elements which are part of such a structure are not represented as direct sub elements any more, instead they are children of the corresponding structure class instance.
This concept enables us to represent arbitrary nested complex structures. In figure 3 the classes are listed which are generated in case of the example element description <! ELEMENT A B, (C|D)>. The content of element A is defined as a sequence of element B and alternatively an element C or an element D, i.e. a choice type. Figure 4 shows how an element, which is structured according to the same definition, is represented with the help of the generated classes listed in figure 3. In this concrete case the choice type is C.

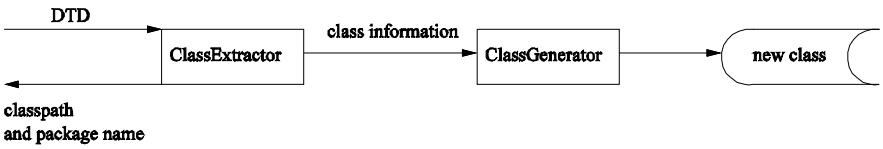
Figure 4.Representation of the structure for the example XML document <A><B/><C/></A> using the classes given in figure 3



Implementation

The class generator XCG consists of two classes called: **ClassExtractor** and **ClassGenerator**. The class extractor extracts any information about elements of a given DTD with the help of the DTD parser implemented by Marc Wutka (Wutka). Parsed structure and type information for classes to generate are passed then to the class generator. This class produces code for any class representing an element or a structure. Right after writing all classes, the class extractor uses the Java compiler to compile the generated Java files and returns the corresponding class path as well as their package name. The whole data flow is shown in figure 5.

Figure 5.Dataflow diagram of our class generator



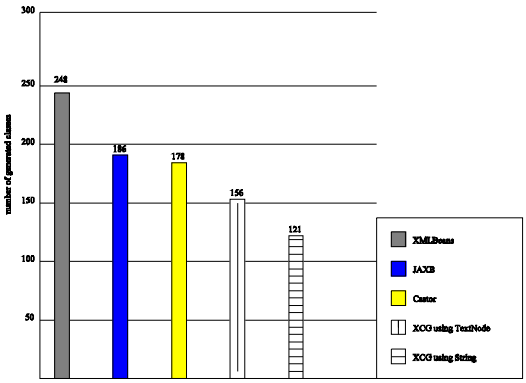
4. EXPERIMENTAL RESULTS

In this section we present some experimental results. For this purpose we use the XMark (CWI,2003) Benchmark test which is designed for XML applications and is supposed to guarantee a kind of standardized testing environment. XMark offers an example DTD and a document generator which is capable of producing arbitrarily large XML documents regarding to the given language description. The size of an XML document can be controlled by a scaling factor. The given XML language description models an auction. The root element is called *site* furthermore there are *persons, open_auctions* and *closed_auctions* elements. An overview of the most important elements and their hierarchy is given in figure 6.
To evaluate the complexity of our new class generator XCG we have taken the class generators described in section 2 and compared the sets of generated classes and interfaces. A large number of generated classes is in conflict with a concise class structure and prevents simple application programming. In this context it is important to notice that text content is represented as *TextNode* by XCG. On the contrary approaches like JAXB and Castor treat text content simply as Java String.

Figure 6 Overview of the most important elements of the XMark auction description.

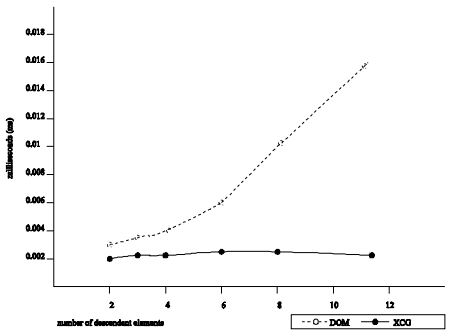
In figure 7 the number of generated class in case of the XMark DTD or an equivalent XML Schema is presented. There are two bars representing XCG our class generator, the first outcomes if textual content is modeled as *TextNode* and the second outcomes if textual content is simply modeled as Java String as it is done by all other approaches.

Figure 7.The number of generated classes for the XMark auction DTD/XML Schema compared with class generators XMLBeans, Castor and JAXB



Another interesting question concerns access times with respect to descendant elements of a given name and or type. For this purpose we have focused on the DOM and our approach. First we have generated an arbitrary XML document corresponding to the XMark XML language description. In a second step these XML documents have been mapped to equivalent Java objects, either a DOM object or an object relying on classes of our class generator. Measured average access times of node test queries retrieving descendant elements of a given type and or name are shown in figure 8.

Figure 8.Average access times for queries retrieving descendant elements of a given type and or name in case of an increasing number of descendant elements



5. FUTURE WORK

Future work will concentrate on extending accepted XML language descriptions to XML Schema in addition to the already supported DTDs. Furthermore it is suitable to normalize DTDs as well as XML Schemas before generating classes. Normalization will prevent dispensable nested element descriptions and thus reduce the number and complexity of generated classes. At the moment XPath is supported for querying instances of the generated classes. Since our model exploits structure descriptions it will be interesting to provide a pattern oriented query language.

6. CONCLUSION

In this paper we have presented our new **X**ML **C**lass **G**enerator (XCG) for XML language descriptions, i.e. DTDs. In contrast to other related approaches like JAXB and Castor, it is now possible to support arbitrary nested and structured element descriptions. In addition structures like lists and unions are explicitly supported by our mapping process. Different from other class generators our class generator provides classes which guarantee lossless mapping. Moreover generated classes build up a clear and simple framework for easily programming applications. At the moment our class generator supports only DTDs, but it will be extended soon to XML Schemas.

REFERENCES

Apache XML Project, T. Xerces Java Parser. http://xml.apache.org/xerces-j/index.html., 2003, Version 2.5

Apache XMLBeans, T. Apache XML Beans. http://xml.apache.org/xmlbeans/index.html. 19. June 2004, Version 2.0

CWI. *XMark – An XML Benchmark Project.* http://xml-benchmark.org, June 2003.

Exolab Group. *The Castor Project*, http://castor.exolab.org/. 2004.

Ort, E. and Mehta, B. *Java Architecture for XML Binding (JAXB).* http://java.sun.com/xml/jaxb/. June 2004.

Sun Microsystems, Inc. *Java 2 Platform, Standard Edition,* v. 1.3.1, API Specification. http://java.sun.com/j2se/1.3/docs/api/index.html. December 2001.

W3Consortium. Document Object Model (DOM) Level 1 Specification, Version 1.0. Recommendation, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/. 1.October 1998.

W3Consortium. *Extensible Markup Language (XML)* 1.0 (Third Edition) W3C Recommendation 04 February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/, February 2004.

W3Consortium. XML Path Language(XPath), Version 1.0. Recommendation, http://www.w3.org/TR/xpath. 16 November 1999.

W3Consortium. XML Schema Part 0: Primer. http://www.w3.org/TR/2001/REC-xmlschema-0-200105002/.Mai 2001.

Inc. Wutka Consulting. *A Java DTD Parser*. http://www.wutka.com/dtdparser.html.

XML About SAX. http://sax.sourceforge.net.