Supporting Set-at-a-time Extensions for XML through DOM

HAI (HELENA) CHEN

School of Computer Science
University of Waterloo
Waterloo, ON
N2L 3G1

CS-2003-27
September 2003

Abstract

With the rapid growth of the web and e-commerce, W3C produced a new standard, XML, as a universal data representation format to facilitate information interchange and integration from heterogeneous systems. In order to process XML, documents need to be parsed and then users can access, manipulate, and retrieve the XML data easily. DOM is one of the major application programming interfaces, which provides the abstract, logical tree structure of an XML document. Though it is a very promising initiative in defining the standard interface for XML documents to access all data required by applications, significant benefit can result if some functions are extended.

In this thesis, we want to support set-at-a-time extensions for XML through DOM. Through extending some powerful functions to get summary information from different groups of related document elements, and filter, extract and transform this information as a sequence of nodes, the extended DOM can reduce the communications overhead and response time between the client and the server and therefore provide applications with more convenience. Our work is to explore the ideas for set-at-a-time processing, define and implement some suitable methods, and code some query application examples by using the DOM and our extensions to make comparisons through a test system. Finally, future work will be given.

Preface to the Technical Report

This technical report reproduces a dissertation presented to the University of Waterloo in 2003 in fulfillment of the thesis requirement for the degree of Master of Mathematics in Computer Science. During the preparation of a subsequent paper,[1] the proposed extensions to the DOM interface were modified slightly as follows:

1. We realized that a major reason for the paucity of operations on a NodeList is so that it can be "*live*, that is, changes to the underlying document structure are reflected in all relevant NodeList … objects." Thus, to preserve this liveness, instead of augmenting the NodeList interface, we suggest that a new NodeSequence interface be introduced with the functionality described in the thesis for NodeLists. For completeness, we suggest one operation to be added to the NodeList interface, namely

   NodeSequence createNodeSequence();

   which will construct a NodeSequence containing the sequence of items included in the object NodeList.

2. To emphasize the mapping of Node operations to operations on each node in a NodeSequence, we suggest renaming the operations described in Section 4.4.2.1 as follows:

Method name	*New suggested name*
getParents	mapParentNode
getFirstChildren	mapFirstChild
getLastChildren	mapLastChild
getPreviousSiblings	mapPreviousSibling
getNextSiblings	mapNextSibling
getChildren	mapChildNodes
getAttributeList	mapAttributes
appendChildren	mapAppendChild
removeChildren	mapRemoveChild
addToChildren	appendChildren
subtractChildren	removeChildren
cloneNodeList	mapCloneNode

[1] Hai Chen and Frank Wm. Tompa, "Set-at-a-time Access to XML through Dom," *ACM Symposium on Document Engineering* (*DocEng'03*), November 20–22, 2003, Grenoble, France.

Acknowledgements

Thanks are given to many people for their help on this thesis. First of all, I would like to thank my supervisor Prof. Frank Tompa, for his guidance, patience and valuable suggestions, especially when I did not know what I was doing. I would also thank my readers Prof. Ken Salam and Prof. Grant Weddell for their great help on modifying this thesis. I am grateful for financial assistance from the Natural Sciences and Engineering Research Council of Canada and from the University of Waterloo. Finally, thanks to my family, who encourage me to face tomorrow.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

With the rapid growth of the web and e-commerce, applications frequently need to process heterogeneous collections of data. To facilitate information interchange and integration from heterogeneous systems, W3C (the World Wide Web Consortium) adopted a new standard, XML (the eXtended Markup Language) [BPS+00]. Since it is a semantics-independent markup language and useful for data that may not conform to a rigid and predefined schema, XML stimulated interest among researchers, and it is being developed by industry as a universal data representation format.

As a unified data format, users should be able to access, manipulate, and retrieve the data conveniently. There are two kinds of APIs (application programming interfaces) to parse XML documents: SAX (Simple API for XML) and DOM (Document Object Model). A SAX parser [Meg02] processes the flow of XML data, detects events when analyzing the tags, and returns event notifications to the application. Since it only provides a simple API to handle XML documents, SAX cannot pass information to applications about nested structure, and it needs more sophisticated implementations to access the data. On the other hand, DOM [ABC+00] represents a parsed XML document as an abstract tree structure that consists of objects. So DOM trees can be directly stored in an object database, and an application can completely access the XML data in any order and update the XML data. Furthermore, restructuring the document via this interface is possible.

DOM provides an object-oriented view of an XML document. Through various interfaces specifying their behaviors, DOM provides the potential to access all data in any order required by applications, but additional functions of DOM could make it more convenient or more efficient to use. For example, applications often need to process collections of related nodes and DOM has limited support for node sets. Thus, much of the work to manipulate node sets is left to applications, and communication overhead between client and server in distributed environments is often proportional to the size of the node set.

The standard DOM interface provides access to the data based on a node interface beginning from the document root. Through this node and by calling various methods, it can traverse each node's children, parent or immediate siblings, one at a time. Though DOM can identify one node's children as a NodeList at one time, it cannot get all the children's children in one operation. Therefore the application programmer must write a loop to process each node in turn. For example, assume an application needs to get all items' names in each region in the sample data from the document named items.xml (with sample data shown in Appendix A). The main section of the source code in C++ and using DOM is shown below:

```
DOM_Document  doc=parser.getDocument();
DOM_NodeList    regions=doc.getDocumentElement().getChildNodes();
unsigned int regionlen=regions.getLength();
for(int i=0; i<regionlen; i++)
{
   items=regions.item(i).getChildNodes();
  unsigned int itemlen=items.getLength();
  for(int j=0; j<itemlen;j++)
      DOM_Node name[i]=items.item(j).getFirstChild();
}
```

In this example, the items that are the children of one region can be accessed at one time using *getChildNodes*, but each item's name must be retrieved iteratively by using the second loop. Often, we need to manipulate all elements of a node set in similar loop, for which it would be useful to express as a single operation. In this thesis we provide some extensions to DOM for operating on collections of nodes.

1.2 Other examples of set-at-a-time processing

Many systems and software, such as SQL, APL, XPath, XQuery, and many others, support the operations for collection of data that do not require a user to iterate through the collection element by element.

Early database systems, such as IMS (Information Management System) [Gel89] and DBTG (Data Base Task Group) [Kor86], provide data operations based on one record or field, and do not support operations on sets. However, the data in a set must often be processed identically, and it is inconvenient for users to use looping structure to iterate over data in a set. Recent database systems including relational (RDBMS) and object-oriented (OODBMS) database mangement systems can process sets of data. In relational database systems [Ull80], many operations, such as selection, projection, set operations (union, intersection, set-difference, cross-product), and join, can manipulate collections of data set-at-a-time. The ObjectStore database system [LLO+91], an OODBMS, improves query operations with more efficiency. It provides a collection facility in the form of an object class library, and therefore a query is simply an expression that operates on one or more collections and produces a collection or a reference to an object.

In a similar way, hypertext databases [DeS86] consist of units of text that are arbitrarily diverse in form and content. Traditional hypertext databases employ a simple directed, labelled model to support editing and traversing the hypertext. In such a model, a unit of information is represented by one node, and nodes are interconnected by links described as a set of labeled, directed and correlated edges. By selecting links, users can only get one node directly through the traditional hypertext interfaces. To obtain sets of nodes, iterative queries are needed and this increases the burden both the user and system. Within the extended hypergraph model [Tom89], users are positioned at arbitrary sets of nodes at one time, and a user has concurrent access to their values or to their neighbours.

Early programming languages such as Fortran and Cobol can only apply functions to one data value at one time. APL (A Programming Language) introduced operations on vectors [Ive62][FaI68]. Typically, a strength of APL is that applying functions to a vector will suffice to process an array of data simultaneously [Pee86]. For example, given a 5-element vector ANGLES that contains the values 30^o, 45^o, 60^o, 90^o, and 120^o, APL applies the *sin* function to the vector ANGLES element by element producing a new vector which contains the values 1/2, $\sqrt{2}/2$, $\sqrt{3}/2$, 1, $\sqrt{3}/2$. This parallel processing is convenient and useful in APL programming.

We also note that W3C produced the XPath [BBC03] and XQuery [BoC03] languages that include operations to construct and combine node sequences to express complex queries, such as concatenate, distinct, sublist, etc. These operations are not provided in the DOM interface. XQuery is designed to

be broadly applicable across different types of XML data sources, and includes many operators, not only on sequences, to make it capable of selecting and extracting complex patterns from XML documents. It is suitable for applications that perform complex manipulations on large XML system. Some of the applications need not use such a powerful and complicated system. We can adopt DOM and add some new functions such as operations on sets to build a simple architecture in Figure1.1 to satisfy the need. Obvious advantages of such a solution are economy, easy of use, and tight integration with the application environment.

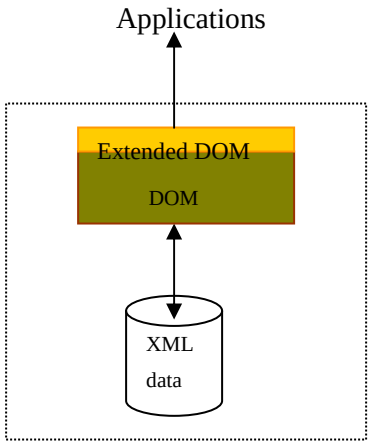


Figure 1.1 The XML architecture based on the extended DOM

These examples illustrate that operations on sets are conveniently used to manipulate information. Therefore, we adopt the idea and recommend our extensions.

1.3 Thesis outline

The purpose of this thesis is to propose set-at-a-time extensions for XML through DOM. To this end, extensions to the NodeList interface are required. Other extensions to the Node interface are also introduced as necessary. We have defined some suitable methods, and we have coded some query application examples by using the standard DOM interface and using our extensions to make comparisons.

Chapter 2 introduces the tree structure of DOM and explains how DOM works. Because the DOM API is specified as a series of CORBA IDL interfaces, we introduce IDL in Chapter 3 and give examples of IDL files and corresponding C++ bindings. Chapter 4 describes our extensions to the

DOM interface, especially the NodeList interface. Beginning with the need for the extensions to the NodeList interface, we give the IDL definition of the NodeList interface, explain our extended methods, and a C++ binding for the extended NodeList interface is shown. Then, implementation considerations are given. Based on our extended DOM interface, application examples are provided in Chapter 5. In the last chapter, we draw conclusions about the advantages of our extensions and outline what remains to be done in the future.

Chapter 2
XML DOM

XML (Extensible Markup Language) [BPM+00], a specification adopted by the W3C, is a popular way to present data in a portable format. In order to process XML, documents need to be parsed by application programs, and then software developers can access, manipulate and manage parsed XML data easily. API tools provide these functions for upper applications, that is, these tools take an XML document and make its structure and content available to an application, often via a standard interface. DOM [ABC+00] is an application programming interface (API) for XML documents defined by W3C. The DOM standard interface can be implemented with any programming language and used in a variety of environments and applications.

Why do we need this interface? As we know, relational database systems have standard interface specifications such as ODBC or JDBC so that database applications need not consider whether the database system is provided by ORACLE, SYBASE or some other vendors. The XML specification only provides the syntax for XML, and, in order to process an XML document to access its content, an XML document needs to be parsed, and components made available to application programs. It would be inconvenient if every parser has a different interface. DOM provides a unified data interface for XML.

2.1 A tree-based API

There are two major different API tools for XML documents: tree-based APIs and event-based APIs. A tree-based API parses an XML document into an internal tree structure, and then allows an application to navigate that tree. Alternatively, an event-based API provides call-backs to application programs whenever significant tokens (e.g. start and end tags) are encountered in the input stream. DOM is a tree-based API, and SAX [Meg02] is an event-based API tool. DOM compiles the entire document directly according to a set of rules provided in the DTD, and constructs a logical tree structure for it in memory (Figure 2.1). XML applications can then process the DOM tree. Most XML parsers produce a DOM representation, and although the interfaces reflect a tree structure, the actual data structure need not be a tree. Documents are modeled using objects that implement:

- interfaces that represent and manipulate a document

- collaborations among these interfaces

A typical DOM tree model for an XML document named “faculty.xml” (Figure 2.2) is represented as in Figure 2.3. When an XML document is loaded and parsed, a DOM tree is produced which includes nearly all the information described by the InfoSet defined by W3C. Accessing from the unique root, the Document node, we can traverse to get different element nodes, attribute nodes, text nodes, comment nodes, and so on. These different types of nodes are presented as objects and describe the XML document’s structure.

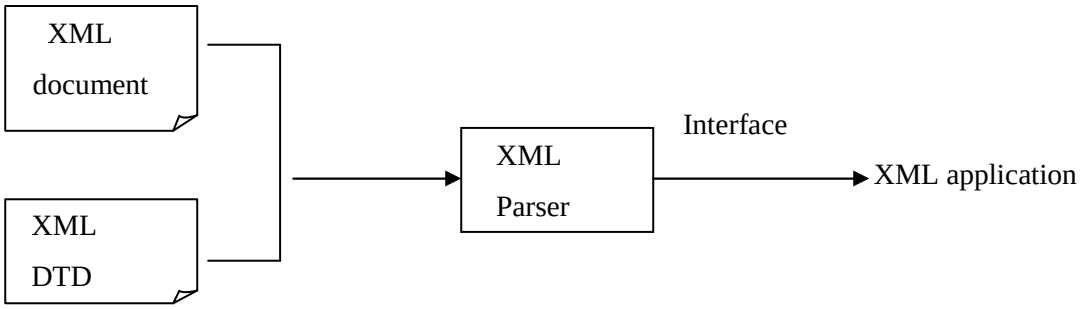


Figure 2.1 The logical graph of XML processing

```
<faculty>
 <person id= “John.Doe”>
  <name><family> Doe</family> <given> John </given></name>
  <email>doe@swen.uwaterloo.ca</email>
  <phone officeroom= “DC3132”>x3128</phone>
 </person>
 <person id= “Tom.Abramov”>
  <name><family>Abramov</family> <given> Tom</given></name>
  <email>abramov@yahoo.com</email>
 </person>
 ….
</faculty>
```

Figure 2.2 A sample instance named faculty.xml

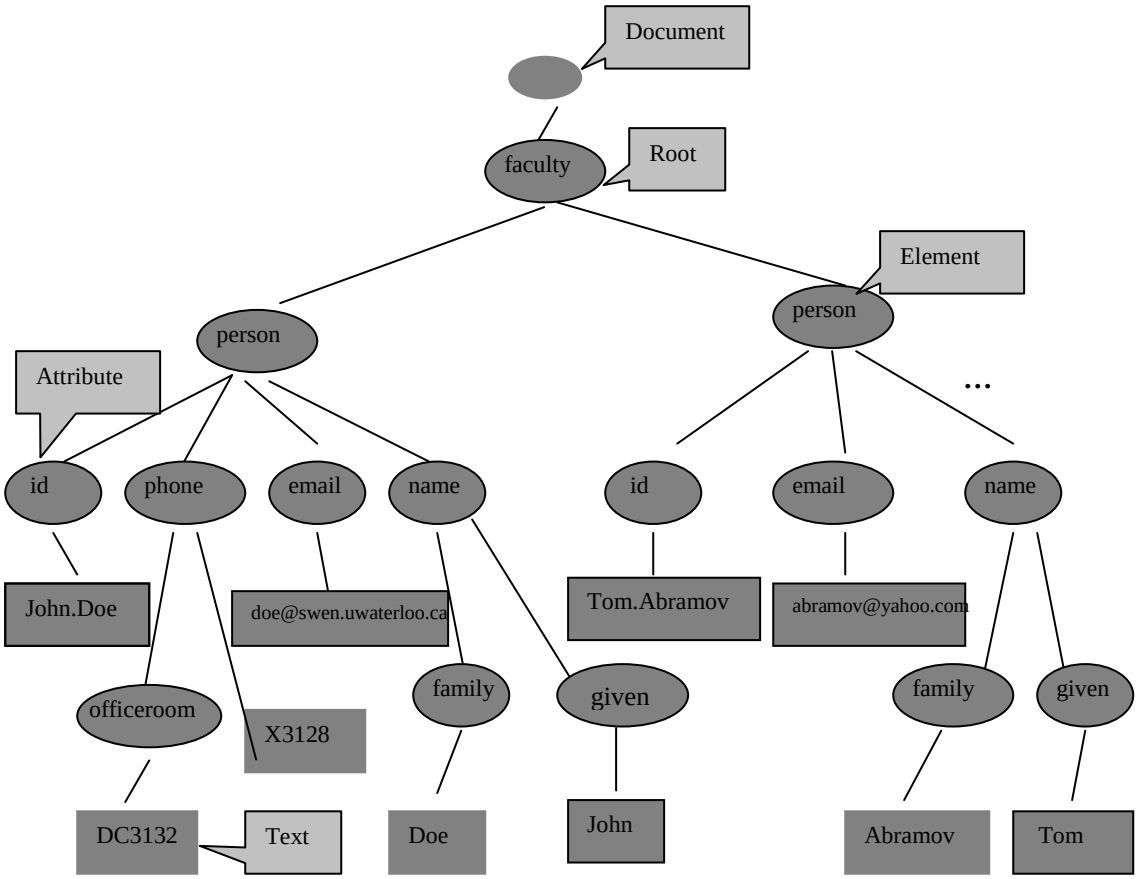


Figure 2.3 The DOM tree structure of an XML document named “faculty.xml”

The fundamental DOM structure model includes the objects represented by interfaces. We can illustrate them by using a simplified UML (Unified Modeling Language) [OMG03] graph (Figure 2.4) which relates these interfaces to each other and can help to understand the relationships among these different interfaces. We can see from the figure that W3C defines twenty-three interfaces in core specifications. Among these interfaces, the Node interface is the primary concept in DOM and the fundamental object, and most important node types inherit from it. Its subclasses include the Element Node, the Document Node, the DocumentType Node, the DocumentFragment Node, the Entity Node, the EntityReference Node, the CharacterData Node, the Attr Node, the Text Node, the Comment Node, the CDATASection, the Notation Node and the ProcessingInstruction Node. (The Text Node and Comment Node inherit from the CharacterData Node directly, so Node is still a

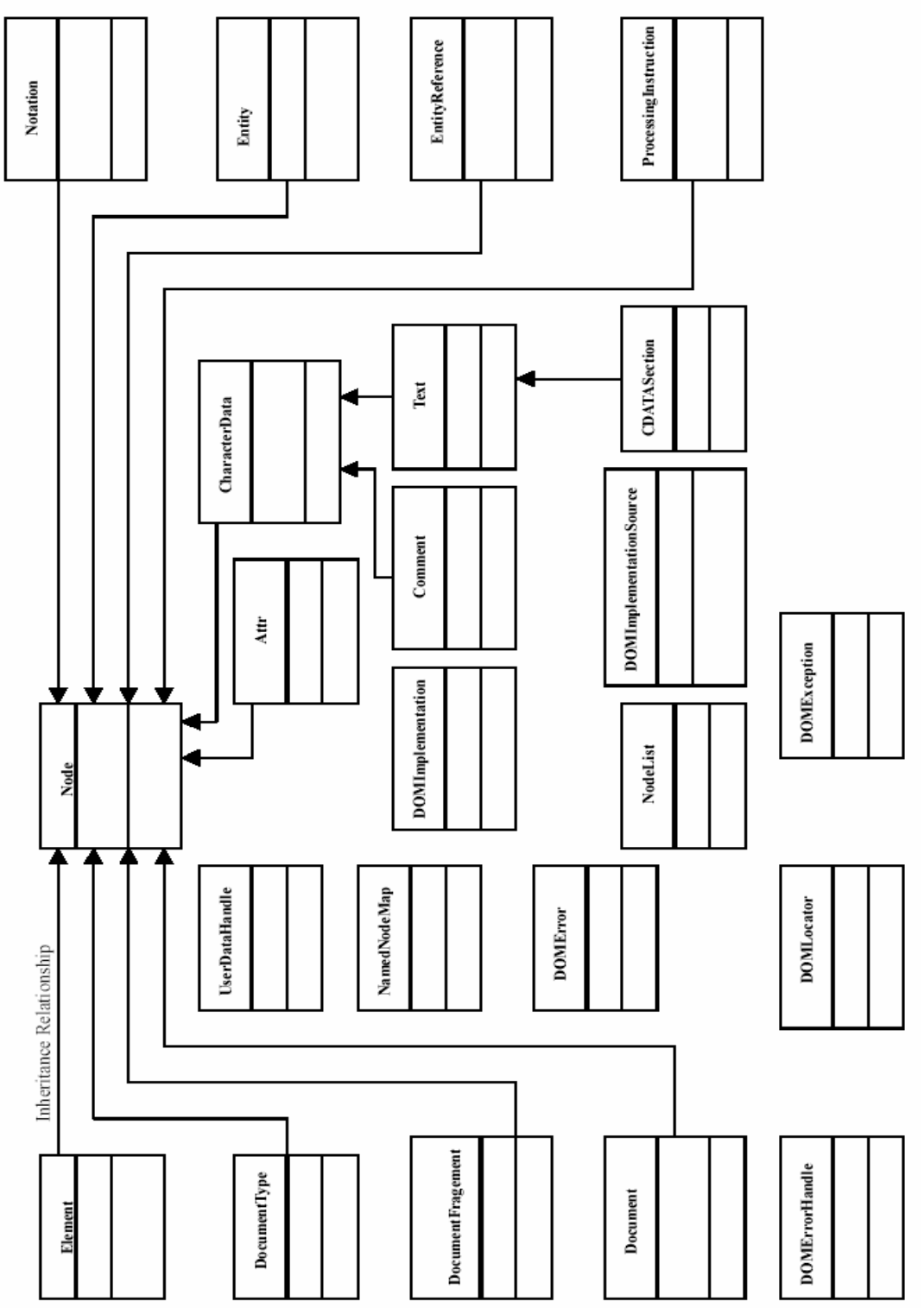


Figure 2.4 The interfaces defined by W3C in DOM core specification

superclass. The CDATASection Node inherits from Text and also inherits from Node indirectly.) The Node and the Character Data Node can't be instantiated. The Node interface defines some methods including appendChild, cloneChild, hasChildNodes, createElement, insertBefore, removeChild, replaceChild, isSupported, compareTreePosition, isSameNode, lookupNamespacePrefix method, and so on.

Figure 2.5 presents some information that an element Node provides. From the element Node interface, we can traverse to find its parentNode, nextSibling, previousSibling, Child NodeList and attributes.

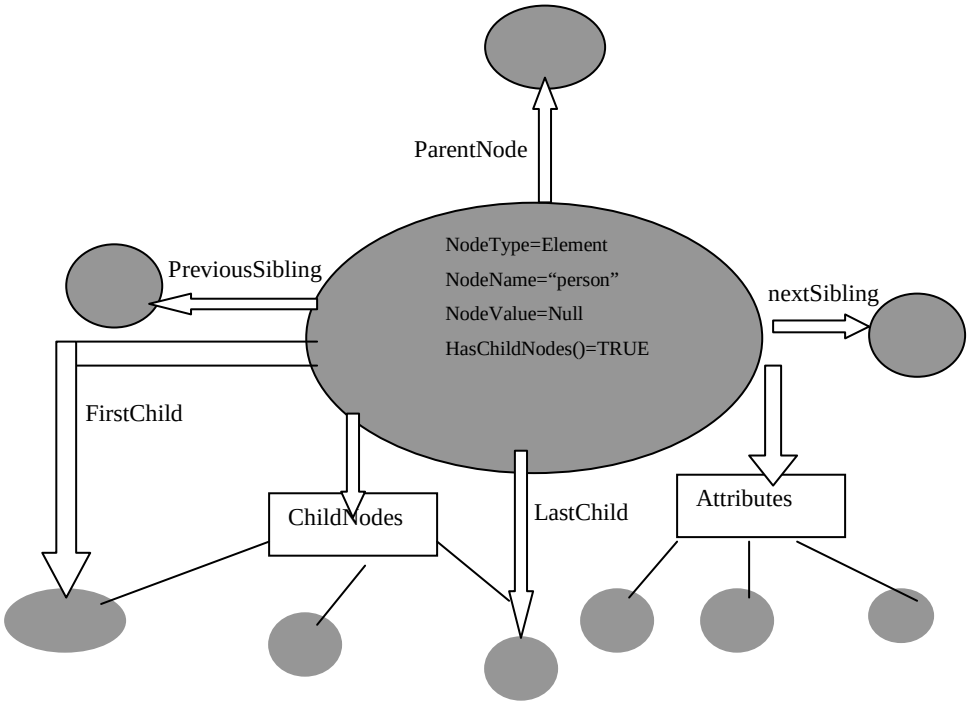


Figure 2.5 An example of an element node in faculty.xml

Another important subtype is Document Node. It is the root of the entire DOM tree and the entry point to access and manipulate an XML document. Every XML document has only one Document Node, and all nodes are descendants of the document node. That is, the Document Node stands for the XML document as a whole, as shown in Figure2.6.

The NodeList object relates a node to its children as part of a node tree. The only attribute for NodeList defined by W3C is the length, which returns the number of nodes in a NodeList, and NodeList has one

method named *item*, which returns a specified node in the NodeList.

The NamedNodeMap object is used to represent the set of attributes, and it is not maintained in any order. It denotes the relationship of a set of nodes and their unique corresponding names.

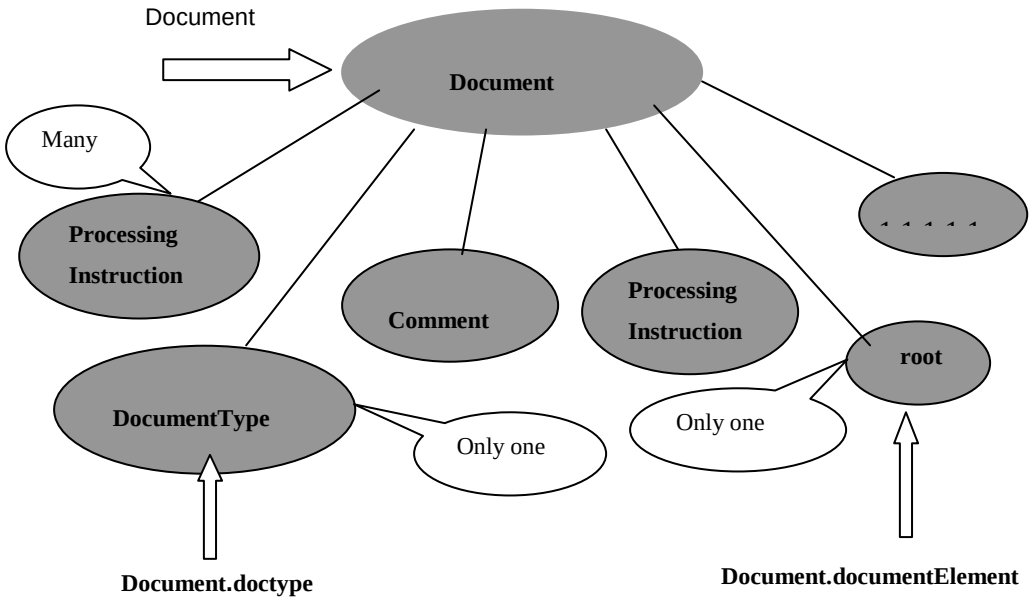


Figure 2.6 A general Document Node

W3C has produced a multi-level specification for DOM. At present, three levels [Heg03] are released by W3C as follows:

1. Level 1 was released in 1998 and supports XML1.0 and HTML. It includes two parts, DOM Core and DOM HTML, and concentrates on the actual core document model, applicable to both XML and HTML, but more specifically HTML. The core functionality provides a low-level set of fundamental interface that can represent any structured document, as well as defining extended interface for representing an XML document. HTML level 1 is an additional, high-level interface.

2. Level 2 was defined in November, 2000, and includes six sections: Core, HTML, Traversal and Range, Views, Events, Style. Based on the Level 1, Level 2 includes some new platform-independent and language-independent interfaces. It supports namespaces, and therefore adds some new methods related to the namespaces in some interfaces defined in core specification. It can also handle document events and adds support for Cascading Style Sheets (CSS).

3. The Level 3 draft became public in September, 2001. It builds on Level 2 and comprises Core, Abstract Schemas and Load and Save, Events and XPath. It provides many functions, such as document loading and saving, content models (DTDs and schemas), key events and event groups.

Table 2.1 lists the composition of different DOM Levels, and Table 2.2 summarizes how Level 2 and Level 3 extend level 1 Core.

DOM Level 1	DOM Level 2	DOM Level 3
Core	Core	Core
HTML	HTML	
	Traversal and Range	Abstract Schemas and Load and Save
	Views	
	Events	Events
	Style	
		Xpath

Table 2.1 The composition of different DOM Levels

Interfaces of Level 1 core	DOM Level 2 extensions based on Level 1 core	DOM Level 3 extensions based on Level 2 core
DOMException	Exception code: INVALID_STATE_ERR, SYNTAX_ERR,INVALID_MODIFICATION_ERR NAMESPACE_ERR, INVALID_ACCESS_ERR	
DOMImplementation	Methods: createDocumentType createDocument	Method: getInterface
		Interface:DOMImplementationSource method :getDOMImplementation
DocumentFragment		
Document	Methods:importNode, createElementNS, createAttributeNS,getElementsByTagNameNS, GetElementById.	Attributes: actualEncoding, encoding, standalone,strictErrorChecking, version Methods: adoptNode, setBaseURI.
Node	Attributes: namespaceURI, prefix, localName Methods: IsSupported, has Attributes, normalize(moved from Element node)	Attributes: baseURI, textContent Methods:compareTreePosition, IsSameNode,lookupNameSpacePrefix, lookupNamespaceURI, normalizeNS,

		isEqualNode, getInterface, setUserData, getUserData
NodeList		
NamedNodeMap	Methods:getNamedItemNS, getNamedItemNS, removeNamedItemNS	
CharacterData		
Attr	Attribute: ownerElement	
Element	Methods: getAttributeNS, setAttributeNS, removeAttributeNS, getAttributeNodeNS, setAttributeNodeNS,getElementsByTagNameNS hasAttribute, hasAttrinuteNS	
Text		Attributes:IsWhitespaceElementContent, wholeText. Methods: replaceWholeText.
Comment		
CDATASection		
DocumenType	Attributes: publicId, systemId, interSubset.	
Notation		
Entity		Attributes: actualEncoding, encoding, version.
EntityReference		
ProcessingInstruction		
		UserDataHandle Interface: Method: handle.
		DOMError: Attributes: Severity, message, exception, location.
		DOMErrorHandler: Method: handleError.
		DOMLocator: Attributes:lineNumber, columnNumber, offset, errorNode, uri.

Table 2.2 How Level 2 and Level 3 extend Level 1 Core.

2.2 Exploitation of DOM

Since W3C defined the fundamental interfaces of DOM in 1998, many implementations of an XML processor have been implemented in a variety of programming environments. DOM parsers provide two services: one is to convert an XML document into a virtual DOM tree, the other is to give some components from the DOM tree to an application. Of those parsers, a popular one is the Xerces C++ parser [Xerces website] which provides the full DOM interface. In order to be portable, minimal use of templates, minimal use of #ifdefs, and no namespaces are included.

In what follows, we use the Xerces C++ parser (open source for Xerces C++ version 1.5.1) as a development platform to explain how DOM works.

2.2.1 Creating a DOM parser to load an XML document

When an XML document is given, we need to navigate its structure, add, modify, or delete its elements using the interfaces defined by W3C. The following example loads the input XML file named “faculty.xml”, invokes the XML4C DOM parser, and then, builds the DOM tree for it. Example 1:

First, we initialize the XML4C system using the following method:

```
try
 {
   XMLPlatformUtils::Initialize();
  }
catch (const XMLException)
  {
cerr << "Error during initialization! :\n"
return 1;
}
```

If initialization fails, catch any exceptions using the *catch ()* method.

Then, we may load the XML document into the parser:

```
    const char*            xmlFile = "faculty.xml";
    DOMParser::ValSchemes  valScheme = DOMParser::Val_Auto;
    bool                   doNamespaces = false;
    bool                   doSchema    = false;

    DOMParser parser;
    parser.setValidationScheme(valScheme);
    parser.setDoNamespaces(doNamespaces);
    parser.setDoSchema(doSchema);
    DOMCountErrorHandler errorHandler;
    parser.setErrorHandler(&errorHandler);


try
        parser.parse(xmlFile);
    catch (const XMLException)
    {
      cerr << "\nError during parsing: '" << xmlFile << "'\n"
      XMLPlatformUtils::Terminate();
      return -1;
     }
  if (errorHandler.getSawErrors())
      cout << "\nErrors occured, no output available\n" ;
  else
  {
     DOM_Document doc = parser.getDocument();
     unsigned int elementCount = doc.getElementsByTagName("*").getLength();
     cout << xmlFile << ": " << elementCount << " elements)." ;
   }
 XMLPlatformUtils::Terminate();
```

In this example, instantiating the DOM parser first requires three methods: *parser.setValidationScheme(valScheme), parser.setDoNamespaces(doNamespaces)* and *parser.setDoSchema(doSchema).* The variable *valScheme* may be set to the DTD, *Val_Always* means the parser should validate the DTD, *Val_Never* means it does not validate the schemes and *Val_auto* means the system selects automatically. When the Boolean variable *doNamespaces* is "TRUE", it enables namespace processing, and *doSchema* enables schema processing. We choose to set both to false for our example. We also install the error handler by method *parser.setErrorHandler(&errorHandler)*. Then we use the method *parser.parse(xmlFile)* to load the XML into the parser to parse it, and after parsing successfully, we extract the DOM tree, count the number of elements and print out the count.

2.2.2 Traversing an XML document by using DOM

If we want to get the content from an XML document, we need to traverse the DOM tree. We continue to use the example XML document named "faculty.xml" to show how to traverse the nodes of the DOM tree.

First, the root element node of the XML document is accessed. The related operation is:

```
DOM_Element faculty = doc.getDocumentElement ();
```

Then, we take the first person item, the element node named *"John.Doe"*, visit it and its child nodes:

```
DOM_Element  firstElem = (DOM_Element &)(faculty.getChildNodes().item(0));
DOM_Element  nameElem =(DOM_Element &)(firstElem.getChildNodes().item(0));
DOM_Element  givenElem =(DOM_Element &)(nameElem.getChildNodes().item(1));
DOM_Element  textNode =(DOM_Element &)(givenElem. getChildNodes().item(0));
DOM_Text  textValue = textNode.getNodeValue();
```

For each object of type Element, the *getChildNodes* method returns an object of type NodeList. The method *item(i)* returns the i^{th} node of the NodeList. After executing the above code, the Text node is retrieved and "John", the value of that node is returned.

For visiting the attribute node, the following code may be used:

```
DOM_Element  attrNode = firstElem.getAttributes().getNamedItem("id");
DOM_Element  attrValue = attrNode.getValue();
```

After executing the instructions above, the Attr node is retrieved and *"John.Doe"*, the value of the text object is returned.

2.2.3 Adding a new node into an XML document

The following example shows how to append an element node named *"mem"* at the end of the child nodes of first *person* node.

```
DOM_Element faculty = doc.getDocumentElement();
DOM_Element  firstElem = faculty.getChildNodes().item(0);
DOM_Element  memElem = doc.createElement("mem");
firstElem.appendChild(memElem);
DOM_Text    memVal = doc.createTextNode("Add a new element");
memElem.appendChild(memVal);
memElem.setAttribute("status", "new");
```

First, we find the first *person* node, create an element node, then append this node to the element node named *"firstElem"*, we then create a text node, and insert it to this new element node. We can also set the attribute for this element node.

The output is as follows:

```
<faculty>
 <person id= "John.Doe">
    <name><family>Doe</family> <given> John </given></name>
    <email>doe@swen.uwaterloo.ca</email>
    <phone officeroom= "DC3132">x3128</phone>
    <mem status="new"> Add a new element </mem>
```

```
</person>
<person id= “Tom.Abramov”>
   <name><family>Abramov</family> <given> Tom</given></name>
   <email>abramov@yahoo.com</email>
</person>
….
</faculty>
```

If we want to insert a new child node before a specific child node, rather than at the end of the child nodes, the following variant could be used:

```
DOM_Element  faculty = doc.getDocumentElement();
DOM_Element  secondElem = (DOM_Element &)(faculty.getChildNodes.item(1));
DOM_Element  emailElem =(DOM_Element &)(secondElem.getChildNodes.item(1));
DOM_Element  memElem = doc.createElement("mem");
secondElem.insertBeforeChild(memElem,emailElem);
    DOM_Text    memVal = doc.createTextNode("Adjunct Professor");
memElem.appendChild(memVal);
memElem. setAttribute("status", "new");
```

The result is as follows:

```
<faculty>
<person id= “John.Doe”>
   <name><family>Doe</family> <given>John</given></name>
   <email>doe@swen.uwaterloo.ca</email>
   <phone officeroom= “DC3132”>x3138</phone>
   <mem status=“new”> Add a new element </mem>
</person>
<person id= “Tom.Abramov”>
   <name><family>Abramov</family> <given> Tom</given></name>
   <mem status=“new”> Adjunct Professor </mem>
   <email>abramov@yahoo.com</email>
```

```
</person>
  ….
</faculty>
```

2.2.4 Modifying an element's content

Modifying an element's content includes changing an attribute, name, and the text content of this node. The following instructions modify the text value of the "mem" element node.

```
DOM_Element  memElem =(DOM_Element &)(firstElem.getChildNodes.item(3));
memElem.childNodes.item(0). setNodeValue("Associate Professor");
```

2.2.5 Deleting a node from an XML document

Now, we delete the element node appended above.

```
DOM_Element    memElem=(DOM_Element &)
                              (firstElem.removeChild(firstElem.childNodes.item(3)));
```

The *"memElem"* node is the deleted one. In deleting this node, the sub-tree whose root is selected will be also deleted.

2.2.6 Creating an XML document by using DOM

We can use DOM to create an XML document directly, and the tree created in this way in memory is the same as the one that the DOMParser would have created.

For this example, after initializing the XML4C system correctly, we create DOM_DOMImplementation object, generate a DOM document whose root element name is "MyRoot" by using the provided method *createDocument(0, "MyRoot", DOM_DocumentType()).*

```
try
   XMLPlatformUtils::Initialize();
catch (const XMLException)
 {
    cerr << "Error during Xerces-c Initialization.\n";
    return 1;
 }
DOM_DOMImplementation impl;
DOM_Document doc = impl.createDocument(0, “MyRoot”, DOM_DocumentType());
```

After using the *getDocumentElement()* method to get the root node, we create an element node *“software”* and append this child to the root node *“MyRoot”*, then a text node *“Xerces-C++”* is generated and inserted to the element node *“software”* as its value.

```
DOM_Element rootElem = doc.getDocumentElement ();
   DOM_Element  softElem = doc.createElement ("software");
   rootElem.appendChild(softElem);

   DOM_Text    softVal = doc.createTextNode ("Xerces-C++");
   prodElem.appendChild (softVal);
```

Following, another child element of the root node is created, an attribute is set, and text content is inserted.

```
DOM_Element  nodeElem = doc.createElement ("MyNode");
   rootElem.appendChild (nodeElem);
```

```
nodeElem.setAttribute ("NodeAttribute", "MyAttributeValue");
DOM_Text    nodeVal = doc.createTextNode ("My Node Value");
nodeElem.appendChild (nodeVal);
```

Next, the last child of root node is created. Now, we have an XML document.

```
DOM_Element  compElem = doc.createElement ("Company");
rootElem.appendChild (compElem);

DOM_Text    compVal = doc.createTextNode ("Apache");
devByElem.appendChild (compVal);
```

The content of the corresponding XML document is as follows:

```
<MyRoot>
  <Software> Xerces-c++</Software>
  <MyNode NodeAttribute=”MyAttributeValue”> MyNodeValue </MyNode>
  <Company> Apache </company>
</MyRoot>
```

Chapter 3
IDL (Interface Definition Language) Specification

As described above, DOM is a language-independent way to access and manipulate XML structures. So, considering using a language-independent system to specify and implement it is natural. CORBA (The Common Object Request Broker) [OMGa02] specifies a system that provides interoperability between objects in a heterogeneous, distributed environment and in a way transparent to the programmer. Its design is based on the OMG Object Model (Object Management Group), and it is fit for specifying well-defined interfaces to access and operate on objects independent of implementation language through OMG IDL. Given such a specification, a specific programming language binding for the IDL specification [OMGa02] implements these interfaces.

In this section, we give a general introduction to IDL, and then illustrate what an IDL file looks like and how it converts to a Java or C++ language binding.

3.1 IDL specification

CORBA, initiated and stipulated by OMG, provides a collection of standards with the aim to reduce the complexity, lower the costs, and facilitate the introduction of new software applications. Objects that conform to this set of standards can interact irrespective of languages, machines, and operating systems. The interface the user sees is completely independent of where the objects are located, in what programming language they are, or any other aspect that is not reflected in the object's interface [OMGb02]. It hides the communication logic, and this ensures the transparency of different locations. For example, when we have a method B of object D implemented in Java and running WinNT, another user can directly call that method in a program implemented in C++ with a different host running on Linux.

The structure of CORBA is object-oriented, and the essential components include OMG IDL, ORB (Object Request Broker), IIOP (a standard communication protocol called the Internet Inter-ORB protocol), BOA (Basic Object Adapter), and stub and skeleton classes. Figure 3.1 shows the composition of CORBA. The heart of CORBA is the ORB, responsible for communicating among the

different objects. That is, it identifies and locates objects, handles connection requests and conveys the arguments from the request to the object implementation through an IDL stub. The crucial part of ORB is the ORB Core, which is accountable for communication of requests. IIOP enables communications among different ORB clients. BOA activates the server objects so that they can receive requests from clients. When these objects are not used any more, BOA will deactivate them. Stub and skeleton classes, which handle the marshaling and unmarshaling of parameters, are automatically generated based on the server objects. They cope with all networking for the client and server objects to make the network transparent to the programmer. In order to keep language-independence and make communications among different objects based on different operating systems and machine environments, there must exist an intermediary, whose function is to map a unified interface description to specific programming languages or object systems. IDL plays such a role, which means all of the interfaces to objects can be defined uniformly and independently of language.

IDL is a language for describing interfaces, and not a programming language. This language defines the types of objects by specifying their interfaces according to the operations that may be performed on them and parameters to those operations [OMGc02]. This means that the interface definition in IDL completely defines the interface and fully specifies each operation's parameters. As long as the implementation of an object corresponds to the definition, it can be programmed in any language. An IDL file corresponding to an object describes the interfaces the object implementations provide. Thus the users know what kind of operations can be used on this object, though they need not know how to implement them.

IDL files facilitate programming. Since different languages need different compilers, and an IDL compiler can compile the IDL file and automatically generate program skeletons for a specific language, programmers only need to fill out the implementation contents based on the skeleton. This is called the language mapping, and the way the IDL is mapped to the programming language is standardized by the OMG. Up to now, the standard mapping languages available include Java, C++, C, Ada, SmallTalk and COBOL.

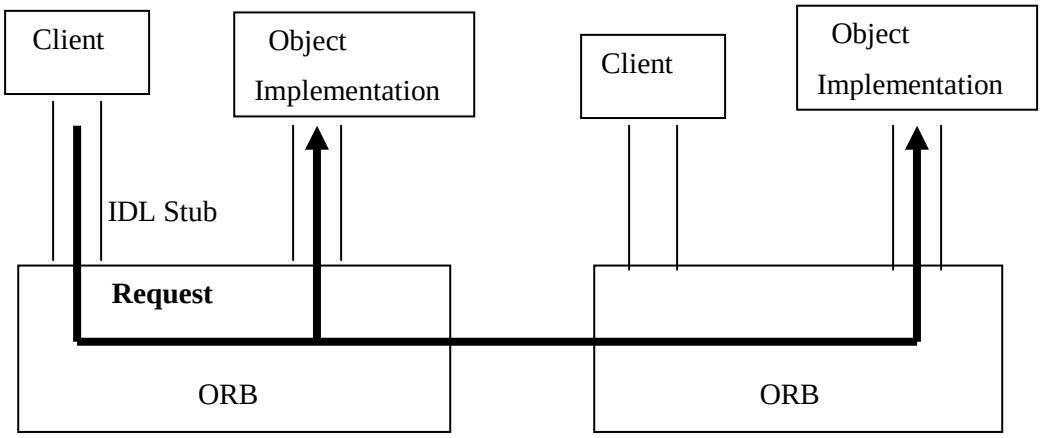


Figure 3.1 The structure of CORBA [OMGb02]

3.2 IDL Properties

As a vendor neutral organization, the W3C does not support a specific platform or operating system. The aim of DOM as a W3C specification is to allow programs to dynamically access and update the content and structure of documents independently of specific platform, so that the users are not forced to adopt a specific operating system. The extension to DOM should similarly not use a specific language in its specification.

Available modeling languages primarily used by distributed systems are IDL, UML (Unified modeling language) and SDL (Specification and description language). These languages can be mapped into several programming languages.

Being sufficiently powerful and widely known, OMG IDL [OMGa03] is a syntactic language that specifies the signature of the interface. It is language-independent and implementation-neutral, and it has defined mappings to several programming languages. Therefore, it provides target language independence between client and server to simplify the interconnection of heterogeneous systems.

As an object-orientated specification language, UML [OMGb03] is a semantic language that specifies, models, and describes the interaction relationship among components of a system. It is easy to understand and learn because it is a higher abstraction that is closer to concepts of humans than IDL, so that UML may be complementary to IDL to help users to understand the relationship between

DOM interfaces more clearly. But UML alone cannot instruct users how to build a system independent of platforms.

As a formal, object-oriented, and graphical specification language to describe how to communicate between systems, SDL [IEC00] is often applied to distributed communicating systems. It is difficult to understand and requires specifications of the communications.

IDL is suitable for modeling new software based on its prominent features as follows.

- Linking objects effectively and conveniently

It is very common that different applications have different requirements. Some require high speed, whereas others require convenience to access the data. These applications, which are not isolated subsystems, may be developed in different programming environments. CORBA acts as a magic glue to allow separated subsystems to communicate with each other through IDL. It defines client and server middleware to extend the space of multiple applications to the entire network so that distributed objects can interoperate together, such as creating and deleting objects, accessing objects, storing them. The entire system is self-describing, with the specification separated from the implementation, so that users can incorporate existing subsystems. IDL can define an object's public interface, so other objects can use it effectively without knowing its internals.

- Programming language independence and transparency among different locations

The DOM describes functionality independently of platforms. Using IDL can make developers know clearly the functionality and philosophy of the DOM. Thus, in distributed systems, on one hand, server developers may use their favorite language to adopt the standard DOM API. On the other hand, client developers can also use the language of their choice to implement application programs based on the DOM API. Figure 3.2 illustrates the architecture in which a client developer creates a query application program using the DOM API as a middleware.

- Portability and deployment

IDL provides a series of separating interfaces from implementation for deployment on a distributed system. In fact, if the system specifies all interfaces using IDL, CORBA can automatically handle many network programming tasks, such as object registration, object activation, and argument

marshaling, so developers need not consider complicated network communication, and extensions and porting are easy to do without changing the original API.

However, IDL is abstract and cannot be used directly to implement distributed applications. API developers must also support a mapping via OMG standards between abstract interfaces and concrete languages befor implementations can be done.

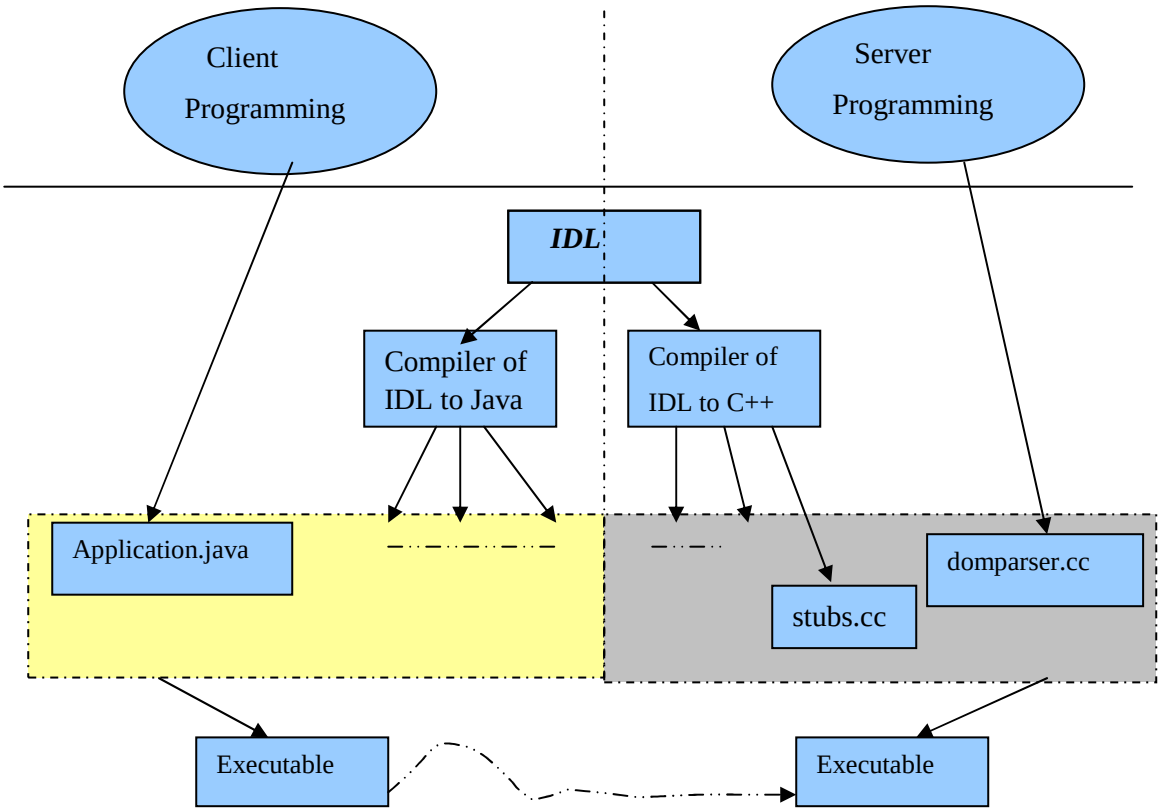


Figure 3.2 Clients develop application program based on DOM APIs the server provides

3.3 IDL syntax and semantics

OMG IDL has its own specific syntax and semantics, though its grammar seems like the C++ language. An IDL specification is composed of one or more type definitions, constant definitions, exception definitions, or module definitions. Its syntax can be described as:

<specification> ::= <definition>+

<definition> ::= <type_dcl>“;” | <const_dcl> “;” | <except_dcl> “;”
| <interface> “;”|<module> “;” | <value> “;”

OMG IDL definitions for W3C’s level3 DOM core definitions, and the IDL files named “dom.idl”, are in http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913/idl.zip. Some sections of dom.idl are as follows:

```
// File: dom.idl

#ifndef _DOM_IDL_
#define _DOM_IDL_

#pragma prefix “w3c.org”
module dom
{
 valuetype DOMString sequence<unsigned short>;
 typedef  unsigned long long DOMTimeStamp;
 …
 interface DOMImplementation;
 interface DocumentType;
 …..
exception DOMException{….}

interface DOMImplementation
 {
  ……
 }

interface Node
 {….}
  ….}
```

Figure 3.3 Some sections of dom.idl

- Module declaration

Now, we give examples of an IDL specification to introduce the more detailed syntax. The complete Module defines a scope, and it is mapped in C++ to the namespace with the same name as a module. According to the IDL grammar, it is described as:

<module> ::= “module ” <identifier>
“{”

<definition>+

"}"

- Interface declaration
  Consider the node interface in dom.idl as shown in Figure 3.4:

```
Interface Node{
  //NodeType
const unsigned short   ELEMENT_NODE= 1;
const unsigned short   ATTRIBUTE_NODE= 2;
  …….

readonly attribute DOMString       nodeName;
attribute DOMString   nodeValue;       // raises(DOMException) on setting
                          // raises(DOMException) on retrieval
readonly attribute unsigned short   nodeType;
…..
Node    insertBefore(in Node newChild, in Node refChild) raises(DOMException);
Node    replaceChild(in Node newChild,in Node oldChild) raises(DOMException);
Node    removeChild(in Node oldChild) raises(DOMException);
Node    appendChild(in Node newChild) raises(DOMException);
boolean  hasChildNodes();
Node    cloneNode(in boolean deep);
void     normalize();
boolean  isSupported(in DOMString feature, in DOMString version);
readonly attribute  DOMString namespaceURI;
attribute  DOMString prefix; // raises(DOMException) on setting
readonly attribute DOMString        localName;
boolean    hasAttributes();
readonly attribute DOMString     baseURI;
……
unsigned short  compareTreePosition(in Node other) raises(DOMException);
```

```
attribute DOMString textContent;  // raises(DOMException) on setting
                        // raises(DOMException) on retrieval
boolean           isSameNode(in Node other);
DOMString         lookupNamespacePrefix(in DOMString namespaceURI);
DOMString         lookupNamespaceURI(in DOMString prefix);
void       normalizeNS();
boolean    isEqualNode(in Node arg, in boolean deep);
Node       getInterface(in DOMString feature);
DOMObject setUserData(in DOMString key,in DOMObject data,in UserDataHandler handler);
DOMObject getUserData(in DOMString key);
  };
```

Figure 3.4 The IDL file of Node interface

An interface, which is similar to a class in C++, is the main component in IDL specifications. It defines the types of data and operations on them, and its syntax is:

<interface> ::= [“abstract”|“local”] “interface” <identifier>
|[“abstract”|“local”] “interface” <identifier> [<interface_inheritance>]
“{” <type_definition>* “;”| <const_definition>* “;”| <except_definition>*“;”
| <attribute_definition>* “;”|<operation_definition>* “}”
<interface_inheritance>::= “:” <interface_name> {“,”< interface_name > }*
<interface_name>::=<scoped_name>
<scoped_name>::=<identifier>| “::” <identifier>|<scoped_name> “::” <identifier>

- Value declaration

  Value type declarations are similar to those in C++. For example, declare DOMString to be a sequence of unsigned short integers.

  valuetype DOMString sequence<unsigned short>;

- Constant declaration

Similarly, Constant Declaration parallels the similar construct in C++. Constants may be of type integer, char, wchar, floating-point, fixed-point, string, wstring and enum, and constant expression is in accordance with the standard definition in C++.

- Type declaration

A common way of naming data types in OML IDL is to use a “typedef” to associate an identifier with a data type through the struct, union, enum, and native declarations. For example,

typedef unsigned long long DOMTimeStamp;

The supported types in OML IDL are listed as follows:

Basic Types	Integer Types	short, long, long long, unsigned short, unsigned long, unsigned long long.
	Floating-point Types	float, double, long double.
	Char Type	char, wchar.
	Boolean Type	boolean.
	Octet Type	octet.
	Any Type	any
Constructed Types	structures	structs, unions, enums
Template Types	Sequence type	sequence
	String Type	string, wstring
	Fixed Type	fixed
Complex Declarator	Arrays Type	multidimensional, fixed-size arrays

Table 3.1 Type declaration in OML IDL

- Exception declaration

Struct-like data structures are used in exception declarations to indicate an exceptional condition that can occur in response to a request. The syntax is:

<except_dcl> ::=”exception” <identifier> “{” <member>*“}”

3.4 C++ language mapping

In general, the mapping information provided by a C++ language mapping includes interfaces, constants, basic data types, enums, types (string, structure, struct, union, sequence, array, typedefs, any), operations and attributes, and arguments. Table 3.2 lists C++ mappings for OML IDL.

IDL type		C++ mapping
Basic Types	short, long, long long, unsigned short, unsigned long, unsigned long long.	short, long, long long, unsigned short, unsigned long, unsigned long long.
	float, double, long double.	float, double, long double.
	char, wchar	char, wchar.
	boolean	bool
	octet	unsigned char
	any	any class
Constructed Types	struct, union, enum	struct, class, enum
Template Types	sequence	class
	string, wstring	char*, wchar_t*
	fixed	Fixed template class
Object	object reference	pointer or object
Attribute	read, write	get method, set method.
interface		class
module		class or namespace

Table 3.2 C++ mappings for OML IDL

The following code listing shows how the IDL file for the Node interface maps to C++.

```
Class Node {
 public:
 unsigned short   ELEMENT_NODE= 1;
 unsigned short   ATTRIBUTE_NODE= 2;
   …….

 DOMString    getnodeName() const;
 DOMString    getnodeValue() const;       // raises(DOMException) on setting
 unsigned short  getnodeType() const;
  …..
 Node         insertBefore(Node newChild, Node refChild) raises(DOMException);
 Node         replaceChild(Node newChild, Node oldChild) raises(DOMException);
 Node         removeChild(Node oldChild) raises(DOMException);
 Node         appendChild(Node newChild) raises(DOMException);
 bool         hasChildNodes();
 Node         cloneNode(bool deep);
 void         normalize();
 bool         isSupported(DOMString feature, DOMString version);
 DOMString    getnamespaceURI() const;
 DOMString    getprefix(); // raises(DOMException) on setting
 DOMString    getlocalName();
 bool         hasAttributes();
 DOMString    getbaseURI() const;
 ……
 unsigned short  compareTreePositio (Node other) raises(DOMException);
 DOMString    gettextContent();  // raises (DOMException) on setting
 bool         isSameNode (Node other);
 DOMString    lookupNamespacePrefix (DOMString namespaceURI);
 DOMString    lookupNamespaceURI ( DOMString prefix);
 void         normalizeNS ();
 bool         isEqualNode ( Node arg, bool deep);
 Node         getInterface (DOMString feature);
 void        *getUserData(DOMString key);
 void        *setUserData (DOMString key, void *data, UserDataHandler handler);
```

Figure 3.5 C++ binding for the IDL file of node interface

Chapter 4
Extensions to the DOM interface

In this chapter, we describe our extensions to the DOM interface, and more specifically to the NodeList interface. All of the examples have been built on the Xerces C++ parser which provides the standard DOM APIs for Windows.

4.1 The need to extend the NodeList Interface

DOM may be considered as middleware to provide applications with the abstract, logical tree structure of XML documents. Though it is a very promising initiative in defining the standard interface for XML documents and many implementations have been developed, significant benefit can result if some functions are extended. XML is flexible in representing many different kinds of information from diverse sources, so the DOM interface should provide features to make applications convenient to retrieve and interpret information from these diverse sources. In particular, although applications often need to process all nodes in a set identically, the NodeList interface provides only limited support for node sets. Some functions must be able to get summary information from different groups of related document elements, and filter, extract and transform this information as a sequence of nodes. More extensions to the NodeList interface can therefore simplify applications, as well as making them more efficient. Two types of methods based on the original NodeList interface are provided:

- Mapping Node operations to every node in a NodeList. For example, *getParents()* extends the method *getParent* for one node to achieve that effect for a set of nodes. That is, these methods operate on each node in the NodeList and produce a sequence of nodes.
- Manipulating the NodeList itself. Such methods include concatenating two sequences of nodes, sorting the sequences of nodes in some order, extracting a section of nodes from the NodeList, constructing a new NodeList, transforming sequences of nodes, eliminating duplicates and so on.

It is apparent that the extensions to the NodeList interface provide applications with more convenience. An example of sample data from the document named price.xml (Appendix B) is illustrated as a simplified DOM tree structure in Figure 4.1. Assume an application operating through a DOM interface needs to count the number of books with different names. The main section of the

source code which reports the length as the count of different elements is given as Example 4.1, where much of the code is for looping over the set of nodes. Example 4.1 is written on top of the DOM API. An alternative solution is shown in Example 4.2, which uses some extensions to the NodeList interface. From the two examples, we can see that the application code can be simplified with these simple extensions.

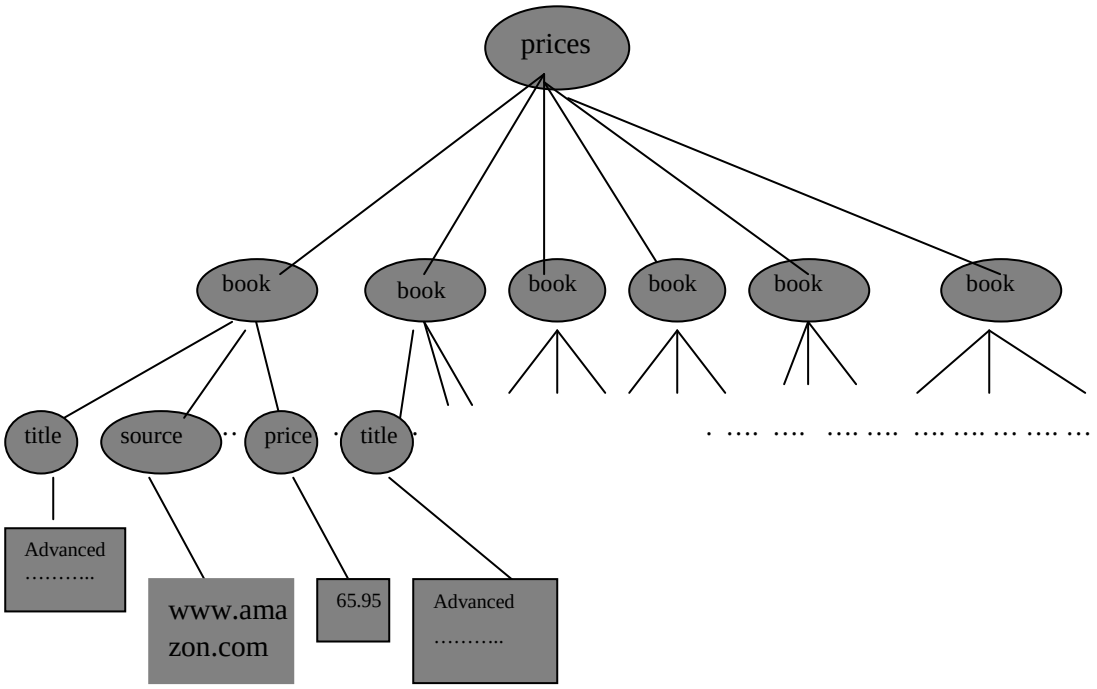


Figure 4.1 The DOM tree structure of price.xml [CFM+01]

l Example 4.1 the main source code by using DOM interface

```
DOM_Document   doc=parser.getDocument();
DOM_Element    rElem=doc.getDocumentElement();
unsigned int      elementCount=rElem.getChildNodes().getLength();
boolean          Duplicate[MAXLEN];
unsigned int      titleCount=0;
for(int i=0;i<elementCount;i++) Duplicate[i]=FALSE;
```

```
for(int i=0;i<elementCount;i++)
  {
     DOM_Node bookElem1=rElem.getChildNodes().item(i);
     DOMString tval1=bookElem1.getChildNodes().item(0).getNodeValue();
     if (Duplicate[i]= =TRUE) continue;
     for (int j=i+1; j< elementCount; j++)
     {
         DOM_Node bookElem2=rElem.getChildNodes().item(j);
         DOMString tval2=bookElem2.getChildNodes().item(0).getNodeValue();
         if(tval1= = tval2) Duplicate[j]= =TRUE;
     }
     titleCount ++;
   }
```

l Example 4.2 the main source code after using the extension to the DOM interface

```
DOM_Document  doc=parser.getDocument();
DOM_Element    rElem=doc.getDocumentElement();
DOM_NodeList   rTitle=rElem.getChildren(1).getFirstChildren(1);
unsigned int        titleCount=rTitle.distinct(true).getLength();
```

Consider next the communication messages back and forth between the client and the server. Let n be the number of books in the given list. In Example 4.1, the number of messages sent through the DOM interface to the server is between 4+3n+3(n-1)=6n+1 and 4+3n+3n(n-1)/2=4+3n(n+1)/2. In Example 4.2, the number of messages is 6. Therefore, the extensions reduce the communications overhead and response time between the client and the server.

4.2 Potential functions from different software language sources

Some languages have defined functions to manipulate sequences which may meet our requirements and can be introduced by our extension interface. We consider here XQuery and XPath operators and APL (A Programming Language).

4.2.1 *XQuery* and *XPath* functions and operators

XQuery and *XPath* functions and operators are designed to be broadly applicable across many types of XML data sources and provide constructors, operators, and functions that are used in XPath and XQuery [MMR+02]. Some useful functions and operators on sequences can be borrowed from it, since in XPath and XQuery a sequence is an ordered collection of zero or more atomic values or nodes, as defined in XML Schema[Fal01][TBM+01] [BiM01], and the type NodeList in the DOM interface is the abstraction of an ordered collection of nodes. These functions are listed below.

- *concatenate*

In XQuery, the concatenate function connects two sequences of items into one sequence. For example, if sequence1 contains items 2,4,5 and sequence2 contains items 3,6,7, then the result of the function returns one sequence (2,4,5,3,6,7).

- *distinct-nodes*

The function distinct-nodes removes all duplicate elements in the given sequence of nodes

- *insert*

This function inserts an element or sequence of elements into a specified position of a sequence. It is a generalization of the appendChild function from the node interface.

- *sublist*

The function sublist(item, start, length) extracts a contiguous sequence of items beginning at the position start and ending at the position start+length-1.

- *intersect*

This function compares two sequences of nodes, item1 and item2, and returns their intersection after eliminating duplicates.

- *except*

The function compares two sequences of nodes, gets the difference of the two sequence arguments and eliminates duplicates. For example, assume two sequences S1={Node1, Node2} and S2={Node2, Node3}, the function except(S1,S2) returns the sequence {Node1}.

- *filter*

The function generates a new sequence of nodes that preserves the original relationship among nodes after excluding intervening nodes that fail to match a given condition.

- *sort*

This function provides a way to reorder items in a sequence.

4.2.2 APL

APL was first designed to be a specification language and later to operate as an interactive language on a variety of computers, including large time-sharing systems, minicomputers and microcomputers. It has been widely used commercially in data processing. The data in APL is structured in arrays. A powerful set of operators are designed to manipulate arrays so that we can expand, compress, and reshape arrays at will. Similarly, in the DOM interface, nodes of the NodeList may be thought of as a vector, the most primitive of arrays. Some functions in APL can also be adopted for the NodeList interface. Here, these functions as follows are helpful with our NodeList interface.

- *catenate*

As for XQuery, the catenate function connects two vectors into a single vector. For example, two vectors as shown below

2 3 4 5, 6 7 8 9

are catenated into the result below (the comma represent the catenate operation).

2 3 4 5 6 7 8 9

- *reshape*

APL uses the *reshape* function⍴to create arrays of any specified shape.

For examples,

Example1: 6 reshape 4 (6⍴4) creates a list of six 4s (4 4 4 4 4 4).

Example2: 5ρ7 9 creates a list of 7 9 7 9 7.

Example3: 5ρ7 8 9 10 11 12 13 creates a list of 7 8 9 10 11.

From the examples, we can see the function will reshape the elements into the shape given. When there are too few elements to fill up the shape, they get repeated (shown as Example 1 and 2). If there are too many elements, only the first ones are used to fill up the shape (shown as Example 3). So using reshape function, we can create a new list, truncate the extra elements or repeat a pattern.

4.3 Design Consideration

The original DOM only provides a tree strucure, and typically returns one node at a time, thus all operations on nodes are left to the applications. Extended DOM produces new methods to process nodes, and new problems need to be considered accordingly.

4.3.1 Return a NodeList with copies or pointers

In the original DOM interface, one node is returned at a time. Whether to treat this node with a value copy or with a pointer is determined by the application. In the extended DOM interface, a collection of nodes is returned at one time. Thus, how to represent these nodes must be considered. For example, when an operation catenates two NodeLists into a new one, one solution is to reference the original nodes with pointers, thus the same nodes can be shared by the new NodeList. An alternative is that we make copies of each node in a NodeList. In most cases, it is preferable to choose the pointer solution rather than value copies based on several considerations:

- Save memory storage

When the data size is large, pointers to nodes will save storage significantly. If we cache these nodes by value, we need to store the duplicates of these node sets. Furthermore, if we use the *cloneNode* method provided by the original DOM, these duplicate nodes will have no parents. Thus, querying of their parents or siblings will be impossible unless we implement a revised copy method.

- Simplicity

Obviously, the return of references is efficient and easy to implement. On the contrary, if we make a copy of each node in a NodeList, we must consider whether it is a shallow copy or deep copy. Furthermore, it complicates update operations. For example, given a collection of nodes in a NodeList, a deletion must remove these nodes from the DOM tree. If we can get the pointers to these nodes, we can remove them directly. However, if we can only obtain a copy of nodes, the

implementation must traverse the DOM tree to match those nodes that are equal to the nodes in a NodeList, and this must be done using a deep equality test.

Of course, when we insert nodes from a NodeList into a DOM tree, we must use a copy of these nodes so that the result is still a tree.

4.3.2 Comparison among nodes

In the extended DOM interface, many operations need to compare collections of nodes such as intersect, distinct, subtract and so on. For example, when an operation intersects two NodeLists, we can choose to remove duplicate objects or duplicate values. Because our NodeLists are represented with pointers, for most type of nodes, such as Element nodes, we compare node sets by their references, that is, we only check whether two nodes are references to the same object. Otherwise, we need to compare the node type, string attributes, and childNodes (perhaps recursively) to check the equality of two nodes. Of course, for text and attributes nodes, comparisons based on values are typically required, such as simple string lookups, chasing references, etc. Thus, we provide a parameter to allow applications to choose whether comparisons are to be done by objects or by values.

4.3.3 The feasibility of apply function

Originally, we intended to adopt the apply function from MAPLE [CGG+85], a language intended specifically for algebraic computation. It introduces a data type, vector, and a function for processing a vector. The map(function, argument) function applies the function to each element in a vector. For example, map(f,{a,b,c}) will yields {f(a),f(b),f(c)}. It simplifies application programs and makes programs more readable.

Similarly, we can also define an *apply* function in the NodeList interface. The result of apply on a NodeList N=(n_1,...n_K) is a NodeList that matches the result of applying the given function with the list of its arguments to each node n_1,...n_K in the object NodeList. Although normally called with a user-defined function, standard functions from the Node interface can also be applied. For example, N.apply("getParents") is identical to forming the NodeList consisting of *n_1.getParent(),...,n_k.getParent(), apply("getChildren")* is identical to *getChildNodes* on the members,

and *apply("removeChildren", children)* is identical to *removeChild* function on each child node of the list of *children*. If the function to be applied to each node takes parameters, then those parameters must be included as additional arguments to apply.

However, implementation of such an apply function is difficult. We have to consider the IDL definition and the result for a C++ mapping. We compare several available design solutions as follows.

Initially, we expect the format for the C++ mapping like this:

NodeList apply (char *funcname, …);

The second parameter of the *apply* function is a type of va_list that for any additional parameters to the function has variable argument list. However, using such an IDL definition is difficult, since va_list is not a fundamental parameter type that IDL can map.

An alternative is to define the IDL interface to include:

NodeList apply (in string *funcname, in string *para);

This requires an application on the client to marshall the variable number of arguments into a string by using the *memcpy* function. Thus, the IDL compiler may pass the string, and the server unmarshall and recover it to the original format by the same function on its end. Obiviously, this solution will bring some inconvenience.

Another alternative is to define *apply* as polymorphic functions according to the number of arguments. Three such *apply* functions appear as follows:

NodeList apply (in string funcname);

NodeList apply(in string funcname, in unsigned short type);

NodeList apply(in string funcname, in NodeList nodes);

This solution will affect the flexibility and scalability of the *apply* function.

Consequently, although the *apply* function could well be useful for the extended DOM interface, it is not adopted in the extended NodeList interface due to this design difficulty.

4.3.4 White space consideration

White space is often used in well-formed and valid XML documents to set apart markup for better readability and clarity, though it is neither data nor markup. Sometimes users turn off validation, and XML parsers must process the whitespace as data. Therefore, our design needs to consider the flexibility of applications to handle new lines. For example, a section of an XML document is shown as Figure 4.2 and the NodeList <1,4> represented by a simplified DOM tree structure is shown as Figure 4.3. However this figure ignores the whitespace(carriage returns and blanks for indentation) that are actually present in the text and therefore also excoded in the DOM tree. Thus a naïve design of the method *getChildren()* would return a NodeList whose length is 8 if it were to behave like the *getChildNodes()* function defined for a single node in DOM. Most users, however, would expect the result to have 3 children only, corresponding to nodes 2, 3, and 5. To avoid such surprising results, a node type parameter is provided in our proposed method *getChildren*. Thus, a call of getChildren(ELEMENT_NODE) will return only nodes of type element and therefore the expected result.

```
<name>
  <surname> Prime </surname>
  <givenname>Tom </givenname>
</name>
<name>
 <givenname> Robus </givenname>
</name>
```

Figure 4.2 A section of an XML document

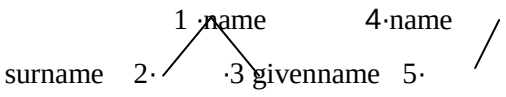


Figure 4.3 Simple tree strucure of a NodeList <1,4>

4.3.5 Liveness consideration

In the original DOM interface, NodeList objects are live, that is, any changes to the underlying document structure are reflected in all relevant NodeList objects. Since our extended NodeList

objects are represented statically in vectors by pointers, the live property will not be preserved. If NodeList is still required to preserve this property, an alternative is that instead of extending the NodeList interface, we can define a new interface, NodeSequence, to put our extention in a NodeList to this interface. In addition, to make the NodeSequence work well, we also define one new method for the Node interface, which makes a node into a NodeSequence, and a second method for the NodeList interface, which makes a NodeList into a NodeSequence.

4.4 Extended NodeList Interface

4.4.1 IDL definition of the NodeList interface

Borrowing ideas from the above examples, we arrive at the following methods for the NodeList interface:

```
Interface NodeList {
    Node        item (in unsigned long index);
    readonly    attribute unsigned long  length;
    NodeList    getParents ();
    NodeList    getChildren (in unsigned short type);
    NodeList    getFirstChildren (in unsigned short type);
    NodeList    getLastChildren (in unsigned short type);
    NodeList    getPreviousSiblings(in unsigned short type);
    NodeList    getNextSiblings(in unsigned short type);
    NodeList    getAttributeList();
    NodeList    appendChildren (in NodeList children);
    NodeList    addToChildren (in NodeList children);
    NodeList    removeChildren (in NodeList children);
    NodeList    subtractFromChildren(in NodeList children);
    NodeList    cloneNodeList(in boolean deep);
    NodeList    catenate (in NodeList newnodelist);
    NodeList    subList (in unsigned long startindex, in unsigned long length);
    NodeList    reshape (in unsigned long num);
    NodeList    subtract (in NodeList delnodelist, in boolean byvalue);
    NodeList    distinct (in boolean byvalue);
```

```
NodeList   intersect (in NodeList othernodelist, in boolean byvalue);
NodeList   sort (in boolean order);
NodeList   filterNodeType (in unsigned short condition);
NodeList   filterTagName(in string Tag);
NodeList   filterValue(in string Value);
}
```

4.4.2 Extended methods of the NodeList interface

We separate the extensions into two types described as above: those that map Node operations to a NodeList and those that support collection operations on NodeLists. Of these operations, *appendChildren()* and *addToChildren()* will create copies of objects, others return NodeLists that contain references of the original objects.

4.4.2.1 Map Node operation to NodeList

4.4.2.1.1 Operations that keep one to one relationship

These operations that return one element always return the element or null. Null is included in the result, so the vectors will remain aligned. It is quite useful when other operations, such as appendChildren(), can process these vectors directly.

- **NodeList getParents ();**

The method returns a sequence of references to the parents, one for each node in the object NodeList. If a node in the NodeList has no parent, the returned value corresponding to that node is null.

Example: To illustrate this operation, consider a simplified DOM tree structure (shown as Figure 4.4). Given a NodeList corresponding to <3,4,6,7>, a call to getParents() produces the NodeList <2,2,φ,6>.

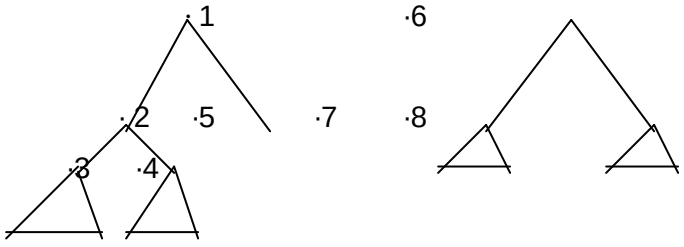


Figure 4.4 A simplified DOM tree structure

(the symbols · denote nodes and triangles represent sub-trees)

- **NodeList getFirstChildren (in unsigned short type);**

type=0, ELEMENT_NODE, ATTRIBUTE_NODE,
TEXT_NODE, CDATA_SECTION_NODE,
ENTITY_REFERENCE_NODE, ENTITY_NODE,
COMMENT_NODE, DOCUMENT_NODE,
DOCUMENT_FRAGMENT_NODE, NOTATION_NODE.

Return a NodeList referencing each node's first child of the specified node type of *type* in this NodeList. The integer indicating the type of node is defined in the original Node interface. The range of *type* is 0 to 10. If type is 0, the first child of each node in this NodeList is returned regardless of type.

Example: Starting again from the state of Figure 4.4 with a NodeList corresponding to <2,5,6>, a call to getFirstChild (ELEMENT_NODE) produces the NodeList <3,φ,7>.

- **NodeList getLastChildren(in unsigned short type);**

Return a NodeList with objects referencing to each node's last child of the specified nodetype of *type* in a NodeList.

Example: Given a NodeList corresponding to <2,5,6>, a call to getLastChild (ELEMENT_NODE) produces the NodeList <4,φ,8>.

- **NodeList getPreviousSiblings(in unsigned short type);**

Return a NodeList with objects referencing to the closest nodes with the node type of *type* that precede each node in a NodeList and matching the type as well.

Example: Starting again from the state of Figure 4.4 with a NodeList which contains <5,8>, a call to getPreviousSiblings(ELEMENT_NODE) produces the NodeList <4, 7>.

- **NodeList getNextSiblings(in unsigned short type);**

Return a NodeList with objects referencing to the closest nodes with the node type of *type* that follow each node in a NodeList and matching the type as well.

Example: Starting again from the state of Figure 4.2 with a NodeList which contains <4,7>, a call to getNextSibling(0) produces the NodeList <5, 8>.

4.4.2.1.2 Operation that keep one to many relationship

These operations on each node of a NodeList return a new NodeList with corresponding nodes. Null node is not included in the returned NodeList.

- **NodeList getChildren (in unsigned short type);**

Return a NodeList that contains references to all of this NodeList's children that have nodetype *type*. As before, if type is 0, all children of this NodeList are returned.

Example: Starting again from the state of Figure 4.4 with a NodeList that contains < 2,5,6>, the result of calling the method getChildren (0) is <3,4,7,8>. Alternatively, getChildren(TEXT_NODE) returns the sublist of {3,4,7,8} that refer to text nodes only.

- **NodeList getAttributeList();**

Return a NodeList with references to the attribute nodes of each node in a NodeList. Though attributes of each node are not maintained in any particular order, an order to these attributes among different nodes is specified according to the sequence number of nodes in a NodeList.

Example: Given a NodeList corresponding to <2,5,6> in Figure 4.5, a call to getAttributeList() produces the NodeList <3, 9, 7>. Since the sequence number of node 6 is larger than that of node 2, attributes of node 2 are put before those of node 7.

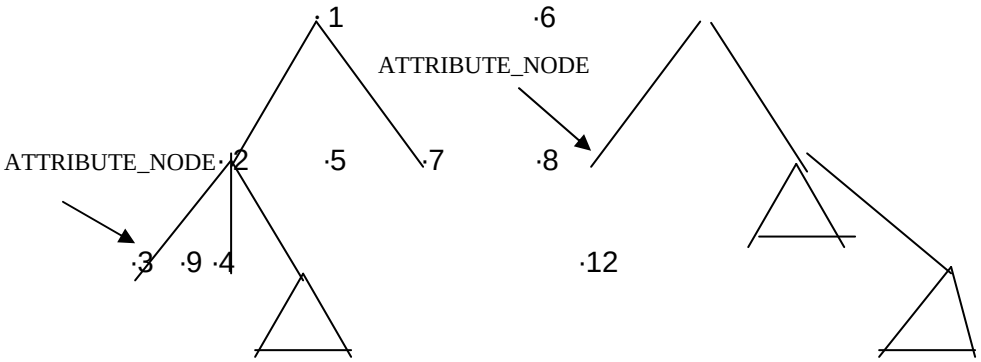


Figure 4.5 A NodeList (2,5,6) represented by a simplified DOM tree

4.4.2.1.3 Update operations of one to one relationship

These operations append or remove children from one NodeList. The number of the NodeList children must be equal to the number of this NodeList, otherwise, an exception is thrown.

- **NodeList appendChildren (in NodeList children);**

Add a copy of the subtree rooted by each node of the NodeList children in order to the end of the list of children for corresponding nodes in this NodeList. The two NodeLists must have the same length, and thus null node is allowed in the NodeList children.

Example: Given a NodeList corresponding to <3,4,7,8> in Figure 4.4 above, and the children NodeList reprensented by the vector <9,10,φ,12>, the result after a call to appendChildren (children) is represented by the Figure 4.6.

- **NodeList removeChildren (in NodeList children);**

Remove each node of the NodeList children from the corresponding lists of children of this NodeList in order. The null node is permitted in the argument NodeList children so that the number of nodes in the children NodeList is the same as the number in this NodeList.

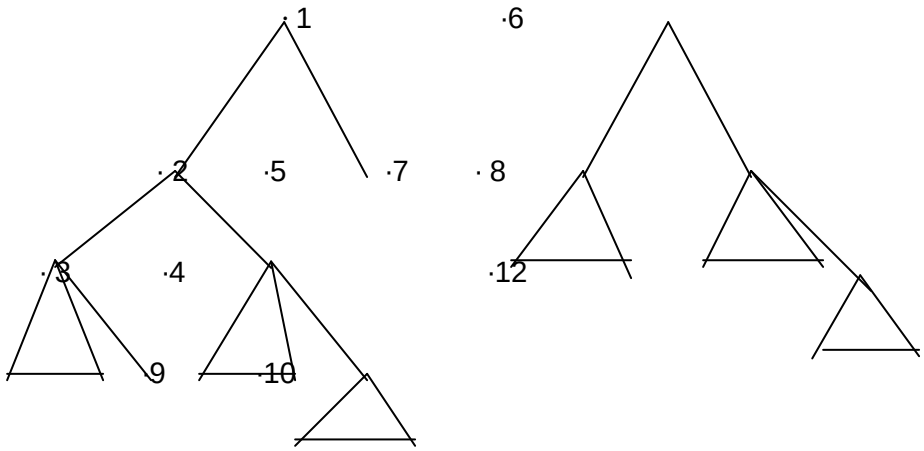


Figure 4.6 The result of a call to appendChild(children)

Example: Consider starting from the status of Figure 4.4. Given a NodeList corresponding to <2,5,6>, and the children NodeList reprensented by the vector <3,φ,8>, the result after a call to removeChild (children) is represented by Figure 4.7.

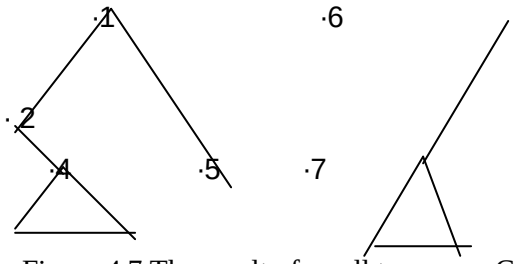


Figure 4.7 The result of a call to removeChild().

4.4.2.1.4 Update operations of one to many relationship

- **NodeList addToChildren (in NodeList children);**

Appends a copy of the complete NodeList children to the end of the list of children for each node in this NodeList.

Example: Given a NodeList corresponding to <3,4,8> in Figure 4.4 above, and the children NodeList reprensented by the vector <9,10,12>, the result after a call to addToChild (children) is represented by the Figure 4.8.

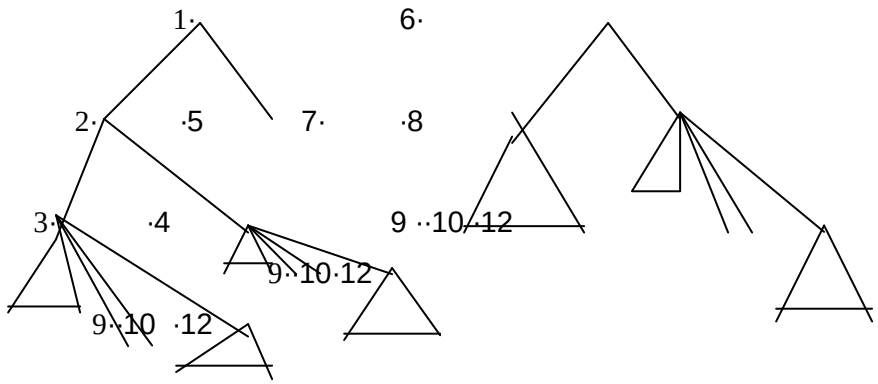


Figure 4.8 The result of a call to addToChild(children).

- **NodeList subtractFromChildren(in NodeList children);**

Remove one or several nodes of the NodeList children from the list of children of this NodeList in order. The null node is not allowed in the NodeList children.

Example: Consider starting from the status of Figure 4.4. Given a NodeList corresponding to <1,6>, and the children NodeList reprensented by the vector <2,5,7>, the result after a call to subtractFromChildren (children) is represented by the Figure 4.9.

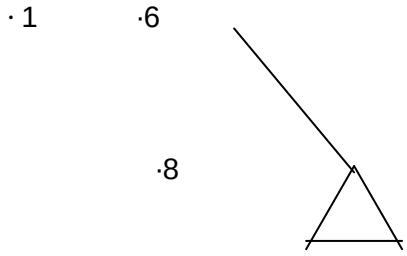


Figure 4.9 The result of a call to subtractFromChildren (children).

4.4.2.1.5 Clone operation

- **NodeList cloneNodeList (in boolean deep);**

Return a new NodeList in which each node is new copy by value of nodes in this NodeList one by one. It also copies all attributes and values. Each copy node in new NodeList has no parents. If it is a deep copy, it copies texts, otherwise, it clones only each node and does not copy any text that each node of this NodeList contains.

4.4.2.2 Collection operations on NodeList

These operations all munipulate NodeLists without regard to their content. Therefore, when the NodeList arguments contain references to nodes, the result will include references to the same nodes.

- **NodeList catenate(in NodeList newnodelist);**

The operation catenates two separate NodeLists into a single NodeList, the result returns a new NodeList consisting of this NodeList followed by a copy of the NodeList newnodelist.
Example: Given a NodeList corresponding to <3,4>, and the newnodelist NodeList represented by <7,8>, the method catenate(newnodelist) returns a new NodeList responding to <3,4,7,8>.

- **NodeList subList(in unsigned long startindex, in unsigned long length);**

Return a section of this NodeList which begins with the given index startindex and ends at the index startindex+length-1. As in the original DOM interface, “the range of valid child node indices is 0 to length-1 inclusive.”
Example: Give a NodeList <3,4,5,7,8>, a call to subList(1,2) will return a NodeList <4,5>.

- **NodeList reshape(in unsigned long num);**

Generate a NodeList of length num by selecting nodes from this NodeList cyclically starting at the first node.
Example: Given a NodeList <3,4,5,6>, the operation reshape(10) will return a new NodeList represented by the vector <3,4,5,6,3,4,5,6,3,4>.

- **NodeList sort (in boolean order);**

Reorder a sequence of nodes to put them into document order (represented by “TRUE”) or reverse document order (represented by “FALSE”) preserving duplicates. Note that if some nodes are not in the base document, for example, they are the cloned nodes and have not been added to the whole tree, we do not reorder node sets.

Example: Given a NodeList corresponding to <2,5,6,4,7,3,5>, a call to sort (TRUE), which means reorder in document order, produces the NodeList <2,3,4,5,5,6,7>.

- **NodeList filterNodeType(in unsigned short type);**

Return a sequence of the nodes that are selected by the type argument from the nodes in this NodeList. If there are no nodes of such node type, this returns null.

Example: Consider starting from the status of Figure 4.10. Given a NodeList corresponding to <2,5,6>, the result after a call to filterNodeType(ELEMENT_NODE) is (2,6).

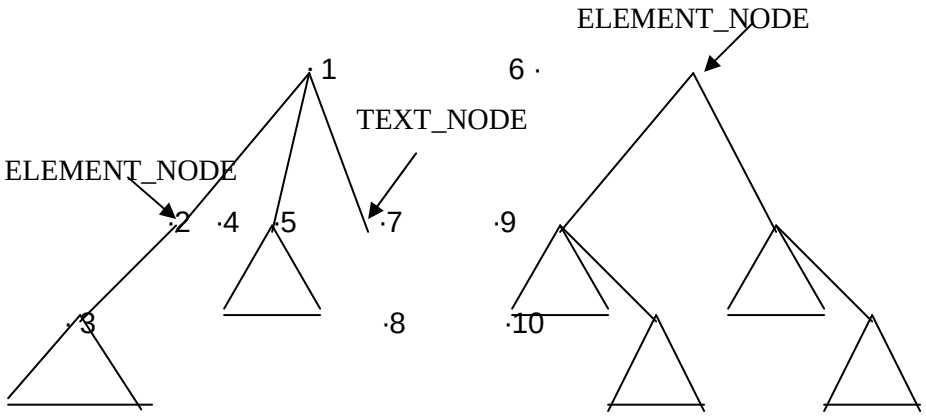


Figure 4.10 A NodeList (1,6) represented by a simplified DOM tree

- **NodeList filterTagName(in string Tag);**

The call to filterTagName(tag) returns a sequence of element nodes whose Tag Name are tag.

Example: Consider starting from the status of Figure 4.11. Given a NodeList corresponding to <2,4,5,7,9>, the result after a call to filterTagName("Title") is <2,7>.

- **NodeList filterValue(in string Value);**

Return a sequence of text nodes that are selected according to the string Value. The call to filterValue (Value) returns a sequence of the nodes whose text value are Value.

Example: Consider starting from the status of Figure 4.12. Given a NodeList corresponding to <2,3,5,7,9>, the result after a call to filterValue ("John.Doe") is <3,9>.

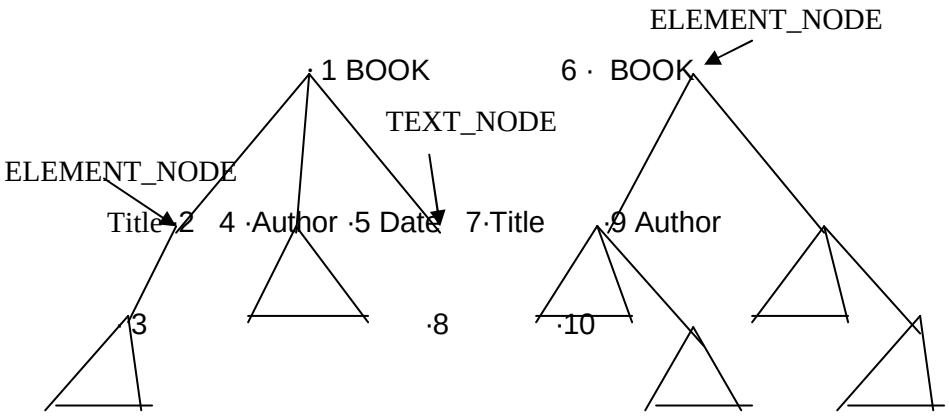


Figure 4.11 A NodeList (1,6) represented by a simplified DOM tree

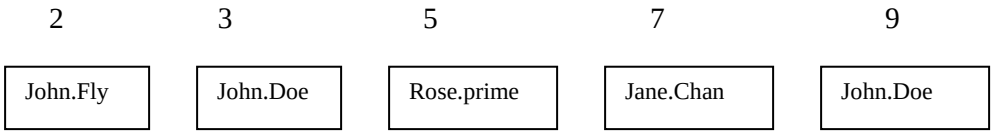


Figure 4.12 A NodeList (2,3,5,7,9) represented by a simplified DOM tree

The following operations need to process text nodes by value.

- **NodeList distinct(in boolean ByValue);**

Eliminate duplicates of this NodeList retaining the first copy in order. Note if ByValue is true, eliminate duplicate text nodes by value, otherwise, eliminate duplicates that reference to the same node.

Example: Given a NodeList <3,4,5,6,3,5>, a call to distinct (false) will return the NodeList <3,4,5,6>. Assume node 5 and node 6 are text nodes and have the same node value, a call to distinct (true) will return the NodeList <3,4,5>.

- **NodeList subtract(in NodeList delnodelist, in boolean ByValue);**

Construct a new NodeList containing the nodes in this NodeList, but not in the argument delnodelist, usually comparing nodes by refence. If ByValue is true, the text nodes are compared by value.

Example: Given a NodeList <3,4,5,6,7>, and the delnodelist <4,7,8,9>, text node 5 and text node 8 have same node value, the operation subtract(delnodelist, true) will return a new NodeList responding to <3,6>. If the delnodelist is null, the operation will return the same NodeList as the original one <3,4,5,6,7>.

- **NodeList intersect(in NodeList othernodelist, in boolean ByValue);**

Construct a new NodeList containing the nodes that appear both in this NodeList and in the othernodelist. If ByValue is true, the values of text nodes are compared, rather than comparing references.

Example: Given a NodeList <3,4,5,6,7>, and the othernodelist <4,7,8,9>, text node 5 and text node 8 have same node value, the operation intersect(othernodelist, true) will return a new NodeList responding to <4,5,7>.

4.4.3 Make some changes for Node interface.

In order to make NodeList operations work well, we also define one new method for the Node interface, which makes a node into a NodeList. Using an argument value greater than one is equivalent to the operation createNodeList(1).reshape(num).

NodeList createNodeList(in unsigned long num);

Return a NodeList which includes num references to this Node.

Example: Given a Node <3>, the call to createNodeList (4) returns a NodeList <3,3,3,3>.

4.5 Contrast to XPath extensions

We note that DOM level 3 makes some extensions to provide simple functionality to access a DOM tree using XPath. The XPath operations build on top of the DOM level 3 Core, and it is intended to unify the DOM and XPath XML data models. It has an ActiveNodeSet interface which has limited operations to process node sets, such as cloneSet(), and getDocumentOrderSet(). However, many capabilities, such as updating and manipulating node sets, are not provided. Futhermore, as XPath converges with XQuery, it is unclear how to preserve the character of DOM while providing XPath functionality.

On the contrary, our extensions build on the existing DOM interface, especially the NodeList interface. It also includes many operations to manipulate node sets, such as mapping node operations to NodeList and manipulating node sets themselves. It complies with the navigational style of DOM without introducing any other one.

4.6 Implementation considerations

Efficient implementation is crucial to the practicality of the extended DOM interface.

4.6.1 Program language considerations

As shown before, CORBA provides programming language independence and supports multiple language mappings for IDL. In addition, IDL defines interfaces independently of implementations. Therefore, our extensions to the DOM can be implemented in different languages. C++ and Java are two popular programming language, and many parsers which support the DOM APIs write in these two language. Industry provides compilers and runtime systems for virtually any platform and operating system. Many parsers faithful to the DOM APIs also implement on top of C++. Xerces-C++ is a validating XML parser written in C++. It provides a shared library for parsing, generating, manipulating and validating XML documents, so it is convenient and reasonable to give our extensions built on the Xerces C++ parser.

4.6.2 Data structure and algorithm consideration

4.6.2.1 Node Vector

In the original DOM interface, we note that the method *getChildNodes()* in the Node interface returns a NodeList. However, that implementation of *getChildNodes()* returns a *NodeList* which caches the parent of these nodes rather than the list of these child nodes. The reason for this approach is that NodeList is designed to be a “live” reference that is updated automatically whenever the underlying DOM tree is updated. The method (i.e. *item()*) applied to such a NodeList can get the corresponding child through the link from parent to child and the link between siblings. Similarly, in the Element Node and the Document interface, the method *getElementsByTagName()*, which initializes a container *node vector* to cache all descendent elements with a given tag name, returns a NodeList which caches the root of the Element tree.

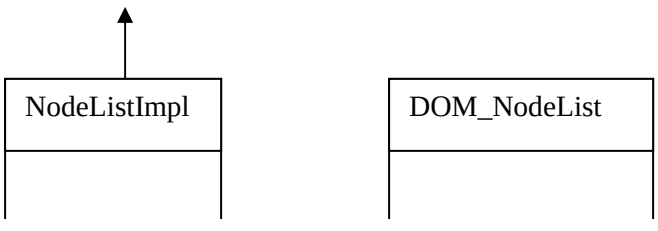
In our extension, because the nodes may have different parents, we cannot use the same approach as *getChildNodes()*. We must return a *NodeList* in which the nodes are explicitly maintained. Hence, a *node vector* storing references to nodes can be applied as a feasible alternative. Accordingly, we need to add new methods for *node vector*. For example, the method *changeElementAt()* swaps two references, and the method *changeElementWith()* overwrites one refrence with another.

4.6.2.2 Sort

The method *sort()* needs to reorder the nodes in document order. Our algorithm needs to traverse the entire tree and record the position of each node of the vector in the tree. For large XML data, the traversal time must be considered. An efficient solution is to cache the result of a traversal and traverse the tree again only if it is modified. For example, when *sort()* is invoked initially, the postion of each node in the tree is cached. When *sort()* is called later, the position of each node in the node vector will be available without re-traversing the tree if it has not changed. Therefore we only need to sort these nodes based on the cached position index.

4.6.3 Interface compatibility

Because our extension is based on the original Xerces implementation, the extended one should be compatible, with as few changes as possible. The hierarchical relationship related to the NodeList interface is shown as Figure 4.13. The internal implementations of the methods in the NodeList interface are in the ParentNode and DeepNodeListImple interface. The reason for two kinds of implementations is that NodeLists are created by different methods, the method *getElementsByTagName()* and the method *getChildNodes()*. However, the extended NodeList interface provides the abstraction of an ordered set of nodes, and all of the implementations need to use node vectors. Thus, all the internal implementations are unified into the *DeepNodeListImpl* interface, and cannot be used directly by application programs. The hierarchical relationship related to the NodeList interface is shown as UML in Figure 4.13.



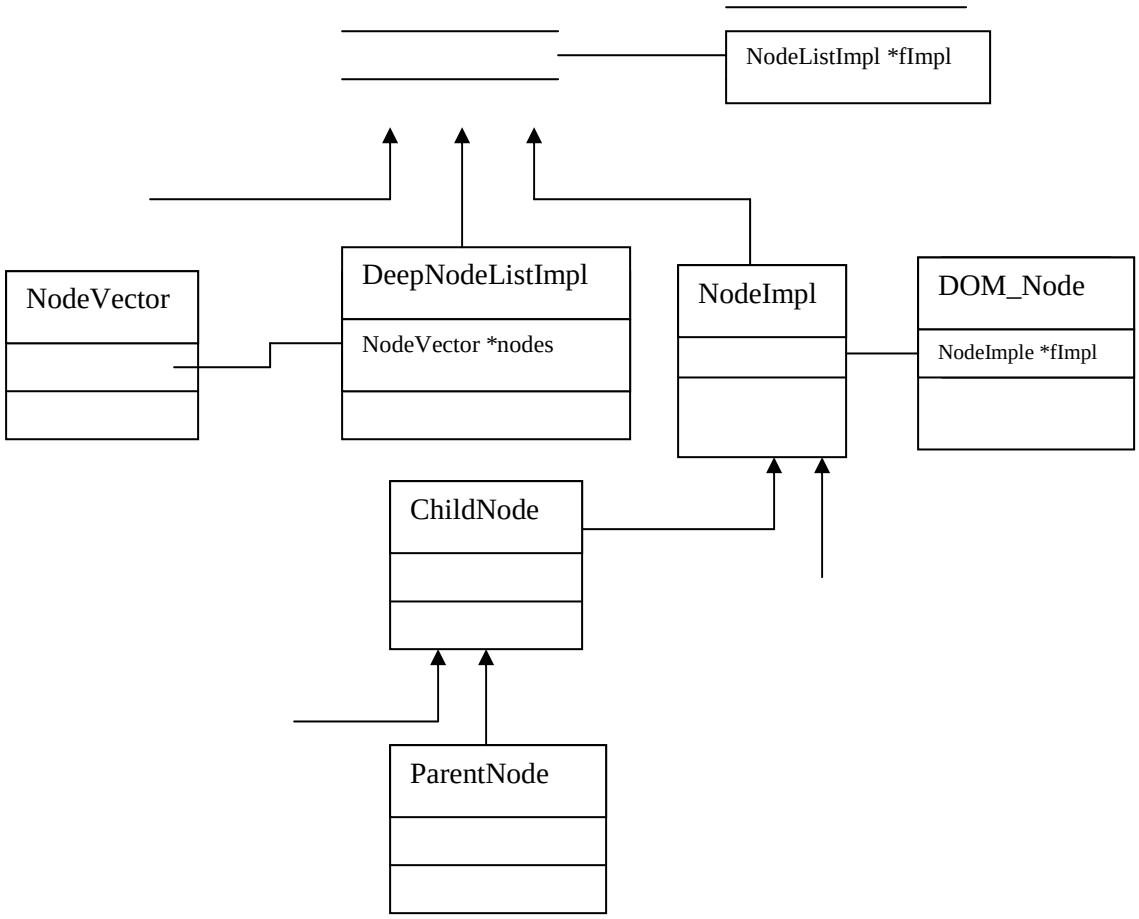


Figure 4.13 The hierarchical relationship related to the NodeList interface

4.6.4 C++ binding for NodeList extension interface

```
Class NodeList {
    Node      item (unsigned long index);
    DOMString  getLength();
    NodeList   getParents ();
    NodeList   getChildren (unsigned short type);
    NodeList   getFirstChildren (unsigned short type);
    NodeList   getLastChildren (unsigned short type);
    NodeList   getPreviousSiblings(unsigned short type);
    NodeList   getNextSiblings(unsigned short type);
    NodeList   getAttributeList();
```

```
    NodeList   appendChildren (NodeList children);
    NodeList   addToChildren (NodeList children);
    NodeList   removeChildren (NodeList children);
    NodeList   subtractFromChildren (NodeList children);
    NodeList   cloneNodeList(bool deep);
    NodeList   catenate (NodeList newNodeList);
    NodeList   subList (unsigned long startindex, unsigned long length);
    NodeList   reshape (unsigned long num);
    NodeList   subtract (NodeList newNodeList, bool byValue);
    NodeList   distinct (bool byValue);
    NodeList   intersect (NodeList newNodeList, bool byValue);
    NodeList   sort (bool order);
    NodeList   filterNodeType (unsigned short condition);
    NodeList   filterTagName(char * Tag);
    NodeList   filterValue(char * Value);
}
```

Chapter 5

Application examples

In the previous chapter, we have given some extensions to the DOM interface and explained how to use them. In this chapter, we illustrate a few practical application examples for the extended DOM interface. Each example includes code and performance data to contrast DOM and extended DOM.

5.1 System environment

One goal of this chapter is to evaluate the performance of the extended DOM. The system is depicted logically in Figure 5.1. The architecture of this system includes two major components:

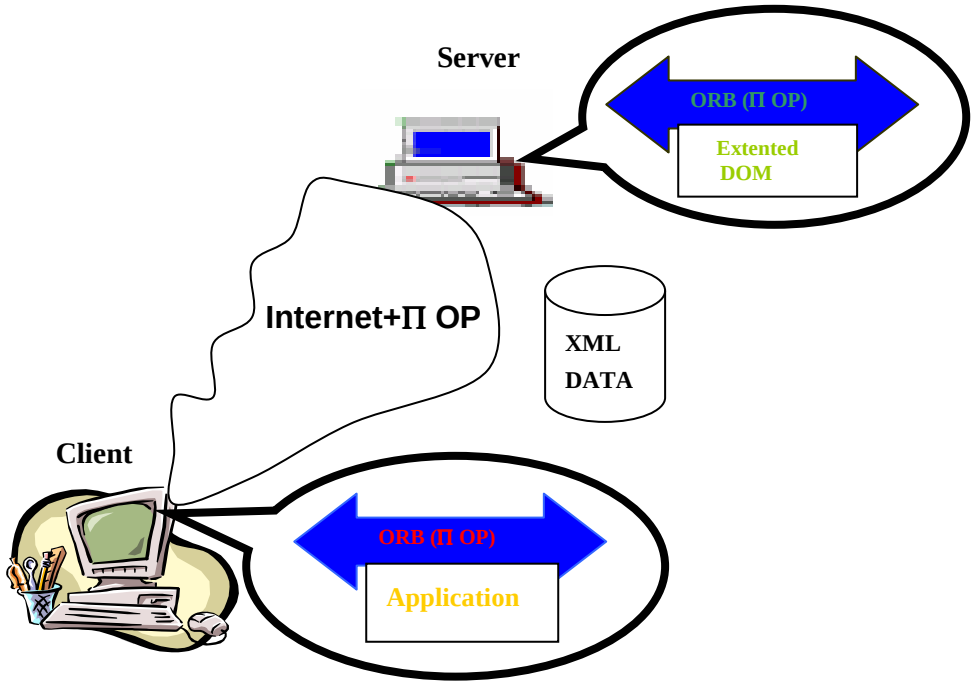


Figure 5.1 Test System Scenarios

- A Server

One possible configuration for the server is Windows2000 with Orbix2000 to provide the CORBA platform. The server must also include the extended DOM interface, and all XML documents are installed on the server.

- Client

The applications are executed in the client. The client calls the extended DOM interface, so the communication messages are conveyed back and forth between client and server.

5.2 XML sample data

With the extensive usage of XML in various application domains, several XML benchmarks have been created to measure, evaluate and optimize the performance of different approaches to deal with XML documents. Since the broad scope of XML makes it difficult to cover all varieties and characteristics of XML data, each family of XML benchmarks can be used to assess specific classes of applications.

The principal XML benchmarks are XMach-1 [RaB02][XMach website]], XMark[SWK+01][XMark website], X007 benchmark[X007 website], Michigan benchmark[MichiganMark website] and Xbench [Xbench website]. Among these benchmarks, the Michigan benchmark is a “micro” benchmark and the data is designed to test basic query operations. Thus the structure of XML data generated is relatively simple and each item is only suitable for a few operations. This makes it inappropriate for our consideration. The Xmach-1 multi-user benchmark is based on a web application and considers different types of XML data, in particular text documents, schema-less data and structured data [BoR01]. It provides hierarchical element structures in each kind of XML document and has few references. The data generated by X007 has few hierarchical element structures. Xbench data is categorized as data-centric and text-centric, but each kind of data has no complex structure that satisfies our requirements. The XMark benchmark represents an auction application combining text and non-text data for items, persons, open auctions, closed auctions, and categories, bidders, sellers, buyers. The relationships between them are expressed through references. We have chosen XMark data for several reasons:

- Variety

We are mainly interested in the primary data characteristics. The XML generator considers the tree fanout, tree depth, and the relationship among different elements through references which distribute in different horizontal and deep level locations of a XML tree. These make XML data complex so it can test our extended DOM’s ability to query efficiently. XMark provides a unified big XML document named “auction.xml” that covers our requirements. Its data structure integrated that of X007 and XMach-1. For example, the structure of the sub-tree rooted at description elements is similar to that of XML data in X007, and its hierarchical structure is similar to that of XML data in XMach-1.

- Utility

The XMark benchmark belongs to the category of application benchmark, and it satisfies four characteristics argued by Gray: scalable (applicable to different size of computer systems), portable (available to implement on different systems), simple (credible) and relevant (performing typical operations for the respective domain) [Gra93]. XMark provides a random data generator to create a single document. The scalable size of the document has wide range from several KB to 10GB. It is apparently simple and portable, since only one big document can present enough complexity for what we need. Finally, it can be used as an input document to measure our extended DOM by application examples for XML processing, and therefore, it is related to our requirements.

The hierarchical schema of the application's XML document named "auction.xml" is depicted in Figure 5.2 and the references that connect sub-trees is given in Figure 5.3 [SWK+01]. "auction.dtd" and a sample data are listed in Appendix C and D respectively.

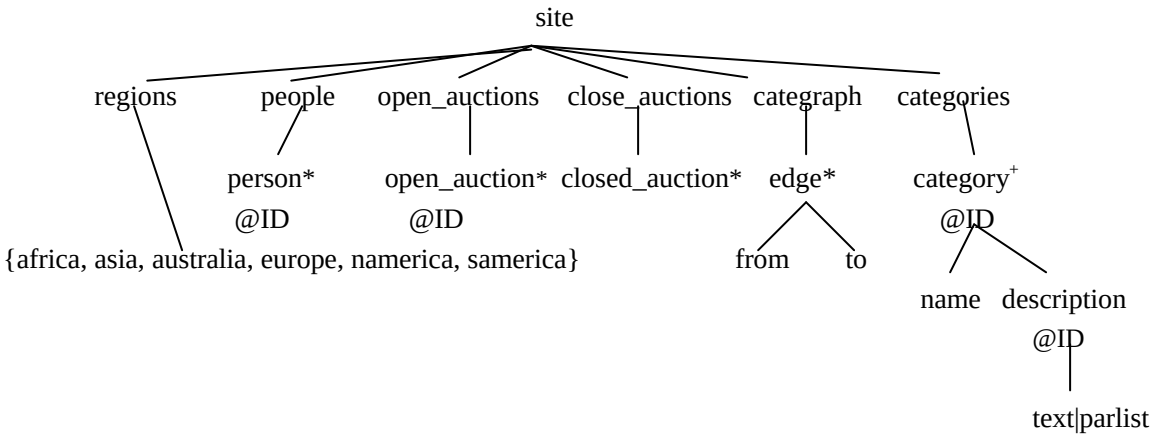


Figure 5.2a Element relationship in site element

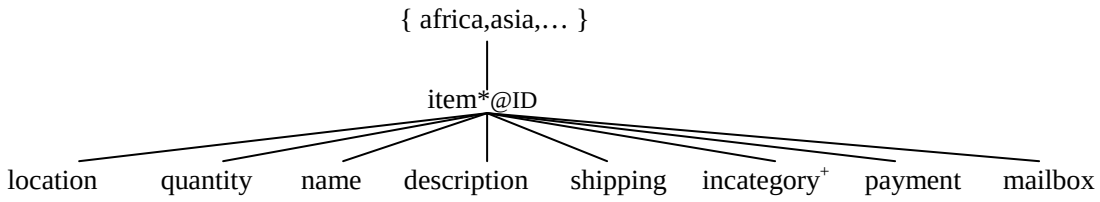


Figure 5.2b Element relationship in regions element

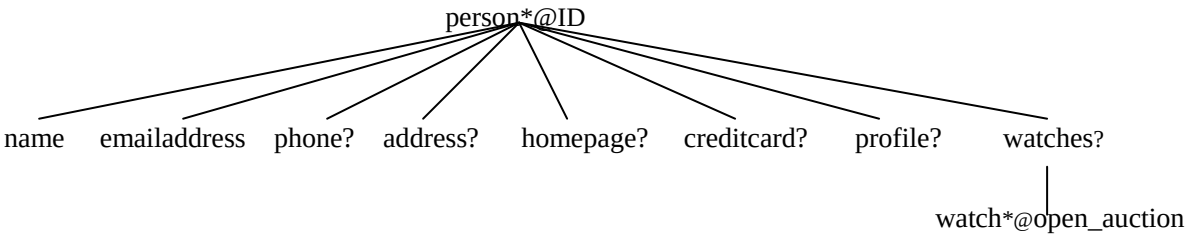


Figure 5.2c Element relationship in person's element

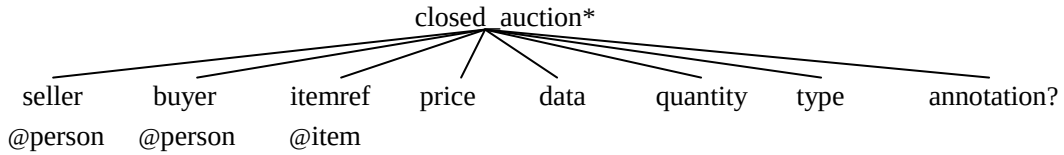


Figure 5.2d Element relationship in closed auction

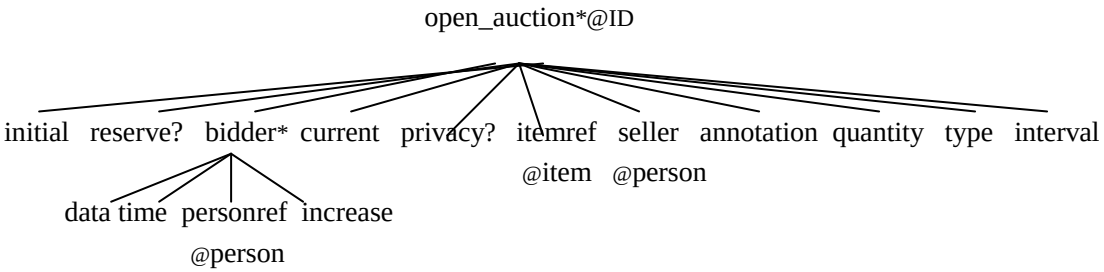


Figure 5.2e Element relationship in open auction

(The symbol + declares that the element prior to it must occur one or more times, the * sign above declares zero or more occurrences of the same element, the ? sign declares zero or one occurrences of the same element, and the | sign declares either/or occurrences of elements.)

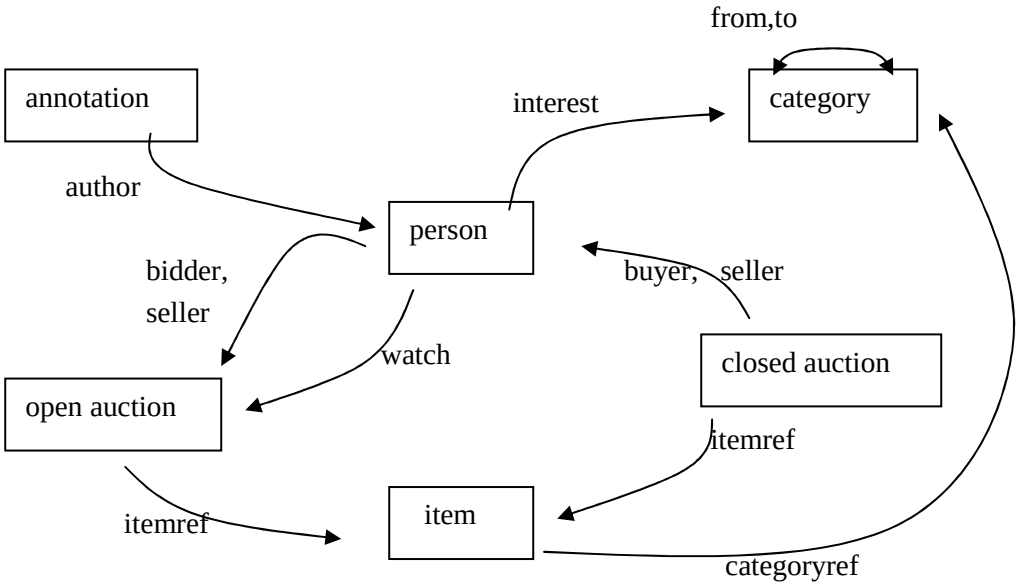


Figure 5.3 References

5.3 Examples for XML processing

We give some examples classified into groups. We begin with simple examples and then provide more complicated ones.

5.3.1 Update a collection of repeated Nodes

The two examples are used to illustrate our extended DOM's ability to remove a collection of nodes from the list of children.

Q1. Delete all descriptions of items.

Note that each item has exactly one description.

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_NodeList items=
doc.getDocumentElement().getElementbyTagName("regions").getChildren(1).getChildren(1);
DOM_NodeList
description=items. getChildren(0).filterNodeType(1).filterTagName("description");
items.removeChildren(description);
```

- Solution using the DOM interfaces.

```
DOM_Document doc = parser.getDocument();
```

```
DOM_NodeList
regions=doc.getDocumentElement().getElementsByTagName("regions").item(0).getChildNodes();
unsigned int regCount=regions.getLength();
for(unsigned int j=0; j<regCount; j++)
{   DOM_Node item= regions.item(j);
    if(item.getNodeType()==ELEMENT_NODE){
    DOM_NodeList items=item.getChildNodes();
    unsigned int itemCount=items.getLength();
    for(unsigned int i=0; i<itemCount; i++)
    {
          DOM_Node item_i=items.item(i);
          if(item_i.getNodeType()==1)
         { DOM_Node
           descript=((DOM_Element &)item_i).getElementsByTagName("description").item(0);
           item_i.removeChild(descript);}
        }
     }
 }
```

The second solution uses two nested loops, and also needs to check whether each node in the inner loop is an element (as opposed to some white space texts). Alternatively, the application based on our extension does not have to use the loop, and it identifies element nodes directly. This greatly simplifies the application code.

Q2. Remove those open auctions whose initial price is 0.

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_NodeList open_auctions=
                    doc.getDocumentElement().getElementbyTagName(“open_auctions”);
DOM_NodeList initials= open_auctions.
        getChildren(0).getChildren(0).filteTagName(“initial”). getChildren(0).filterValue("0");
DOM_NodeList open_auction= initials.getParents().getParents();
```

```
open_auctions.subtractFromChildren(open_auction);
```

- Solution using the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_NodeList open_auction= doc.getDocumentElement().
                    getElementbyTagName("open_auctions").item(0).getChildNodes();
unsigned int i=0;
while((i< open_auction.getLength())&&(i>=0))
 {
    DOM_Element openElem= open_auction.item(i);
    if(openElem.getNodeType()==DOM_ELEMENT)
     {
       DOMString  initial=itemElem.getElementbyTagName("initial").item(0).
                              getChildeNodes().item(0). getFirstChild().getNodeValue();
        if(initial.equals("0"))
          {open_auctions.removeChild(openElem); i--;}
        i++;
     }
  }
```

The second solution gets each node *openElem* by a while loop and checks whether the text value of the node *initial*, a child of *openElem*, is 0. The first one invokes the method *getChildren(0)* to get child nodes of all *open_auction* nodes. The method *filterTagName()*, *getChildren()*, and *filterValue()* are subsequently invoked, and the set of nodes whose value are 0 is returned.

Similarly, the extended DOM provides the methods *appendChildren()* to append each node of a NodeList as a child of the node with the corresponding position in another list and *addToChildren()* to append all the nodes in a NodeList as children of each node in another one..

5.3.2 Select a collection of nodes that have some common structure

The task here is to perform an exact match against a collection of repeated nodes based on a repeated feature such as tag name.

Q3. Get all intervals in all open auctions.

Each open auction has one *initial* element.

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_Element   siteElem=doc.getDocumentElement();
DOM_NodeList   intervals= siteElem.getElementbyTagName(“open_auctions”).
                                getChildren(1).getLastChildren(1);
```

- Solution using the DOM interface

```
DOM_Element interval[MAXLENGTH];
DOM_Document doc=parser.getDocument();
DOM_NodeList   open_auction= doc.getDocumentElement().
                        getElementbyTagName(“open_auctions”).item(0).getChildNodes();
unsigned int openCount= open_auction.getLength();
for(int i=0;i<openCount;i++) {
      DOM_Node openElem=open_auction.item(i);
      if(openElem.getNodeType()==1) {
           interval[j++]=(DOM_Element &)(((DOM_Element &)openElem).
                        getElementsByTagName("interval").item(0));}}
```

To return a collection of node sets, the second solution has to define a structure to store these nodes. The result of the first one is a NodeList which is implemented by the extension. Therefore, it reduces applications’ involvement from coding data structure to storing nodes.

Q4. Get names of items.

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_NodeList nameList=doc.getDocumentElement().getElementsByTagName("regions").
                        getChildren(1).getChildren(1). getChildren(1).filterTagName("name");
```

- Solution using the DOM interface

```
DOM_Element nameElem[MAXLENGTH];
```

```
DOM_Document doc=parser.getDocument();
DOM_ NodeList regions= doc.getDocumentElement().
                    getElementsByTagName("regions").item(0).getChildNodes();
unsigned int regCount= regions.getLength();
for(unsigned int i=0;i<regCount; i++)
 {
      DOM_Node regElem=regions.item(i);
      if(regElem.getNodeType()==1){
      DOM_NodeList items= regElem.getChildNodes();
      unsigned int itemCount= items.getLength();
      for(unsigned int j=0;j<itemCount;j++)
      {
         DOM_Node itemElem=items.item(j);
        if(itemElem.getNodeType()==1)
         nameElem[k++]=(DOM_Element &)(((DOM_Element &)itemElem).
                          getElementsByTagName("name").item(0));
       }
  }
```

The first solution simplifies the implementation from complex loops in the second one to a nested method invocation. The descendents of node *regions* can be obtained by invoking *getChildren()*. The expected results with tag name of "*name*" can be returned by invoking the method *filterTagName()* as the last step.

5.3.3 Select a collection based on value.

These examples also illustrate the extended DOM to get the ancestors and descendents of collections conveniently.

Q5 Count the number of bidders whose id are "person0".

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_NodeList open_auction=
        doc.getDocumentElement.getElementbyTagName("open_auctions").getChildren(1);
```

```
DOM_NodeList bidders= open_auction.getChildren(1).filterTagName("bidder");
DOM_NodeList
     personAttr= bidders.getChildren(1).filterTagName("personref").getAttributeList();
unsigned int perCount= personAttr.filterValue("person0").getLength();
```

- Solution using the DOM interface.

```
 DOM_Document doc=parser.getDocument();
 DOM_NodeList opens=doc.getDocumentElement().getElementsByTagName("open_auctions").
                              item(0). getChildNodes();
unsigned int openCount = opens.getLength();
for(unsigned int i=0;i<openCount;i++)
{
 DOM_Node openElem=opens.item(i);
 if(openElem.getNodeType()==1)
 DOM_NodeList bidders=((DOM_Element &)openElem).getElementsByTagName("bidder");
  unsigned int bidCount=bidders.getLength();
  for(unsigned int j=0;j<bidCount;j++)
  {
    DOM_Node bidElem=bidders.item(j);
    DOMString person=((DOM_Element &)bidElem).
    getElementsByTagName("personref").item(0).getAttributes().item(0).getNodeValue();
    if(person.equals("person0"))   bidsNum++;
  }
}
```

The second solution needs to count the number of nodes of *person*. The first one can get the result by an invocation of the method *getLength()*.

Q6 List those open auctions whose bidder's id is "person0".

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc = parser.getDocument();
DOM_NodeList open_auction=doc.getDocumentElement().
```

```
                                    getElementsByTagName("open_auctions").getChildren(1);
DOM_NodeList bidders=open_auction.getChildren(1).filterTagName("bidder");
DOM_NodeList personAttr=bidders.getChildren(1).filterTagName("personref").getAttributeList();
DOM_NodeList person0=personAttr.filterValue("person0");
DOM_NodeList opens=person0.getParents().getParents().getParents().distinct(true);
```

- Solution using the DOM interface

```
DOM_Document doc=parser.getDocument();
DOM_NodeList opens=doc.getDocumentElement().getElementsByTagName("open_auctions").
                        item(0).getChildNodes();
unsigned int openCount = opens.getLength();
DOM_Element auction[MAXLENGTH];
for(unsigned int i=0;i<openCount;i++)
{
   DOM_Node openElem=opens.item(i);
   if(openElem.getNodeType()==1)
  {
    DOM_NodeList bidders=((DOM_Element &)openElem).getElementsByTagName("bidder");
    unsigned int bidCount=bidders.getLength();
    for(unsigned int j=0;j<bidCount;j++)
    {
      DOM_Node bidElem=bidders.item(j);
      DOMString person=((DOM_Element &)bidElem).getElementsByTagName("personref").
                        item(0).getAttributes().item(0).getNodeValue();
      if(person.equals("person0")){
                                    auction[k++]=(DOM_Element &)openElem;
                                    break;}
    }
  }
}
```

In addition to using complex iterations to descend down the tree to search for the specific text nodes, the second solution has to deal with duplicate nodes. It is notable that the first one only invokes one method *distinct()* to remove duplicates.

Similarly, by invoking methods provided by the extension, such as *getPreviousSiblings()*, *getNextSiblings()*, *getFirstChildren()*, *getLastChildren()*, applications can obtain the corresponding siblings and children of a given node set at one time.

5.3.4 Selection via a join.

Join operations are often useful to evaluate a query. These queries show that the extended DOM can handle element structures with complex relationships.

Q7 Get those bidders' name nodes in the private open auctions. (join person and open_auctions)

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_NodeList   siteChildren=doc.getDocumentElement().getChildNodes();
DOM_NodeList personNodes= siteChildren.filterTagName("people").getChildren(1);
DOM_NodeList idTexts= personNodes.getAttributeList().getChildren(0);
DOM_NodeList openNodes=siteChildren.filterTagName("open_auctions").getChildren(1);
DOM_NodeList privacyNodes=openNodes.getChildren(1).
                    filterTagName("privacy").getChildren(0).filterValue("Yes").getParents();
DOM_NodeList opens=privacyNodes.getParents().distinct(true);
DOM_NodeList bidderAttrs=opens.getChildren(1).
     filterTagName("bidder").getChildren(1).filterTagName("personref").getAttributeList();
DOM_NodeList bidderTexts=bidderAttrs.getChildren(0).distinct(true);
DOM_NodeList ids=idTexts.intersect(bidderTexts);
DOM_NodeList biddersName=ids.getParents().getParents().getChildren(1).filterTagName("name");
```

- Solution using the DOM interface

```
DOM_Document doc=parser.getDocument();
DOM_Element siteElem= doc.getDocumentElement();
DOM_NodeList pers=siteElem.getElementsByTagName("people").
                        item(0).getChildNodes();
```

```
 unsigned int persCount = pers.getLength();
 DOM_NodeList opens=siteElem.getElementsByTagName("open_auctions").
                              item(0).getChildNodes();
unsigned int opensCount = opens.getLength();
DOM_Element name[MAXLENGTH];
bool mark[MAXLENGTH];
for(unsigned int n=0;n<MAXLENGTH;n++) mark[n]=false;
for(unsigned int i=0;i<opensCount;i++){
  DOM_Node openElem=opens.item(i);
  if(openElem.getNodeType()==1)
 {
     DOM_NodeList privs=((DOM_Element &)openElem).getElementsByTagName("privacy");
     if(privs.getLength()>0) {
        DOM_Node privElem=privs.item(0);
        if(privElem.getFirstChild().getNodeValue().equals("Yes")){
         DOM_NodeList bidders=((DOM_Element &)openElem).getElementsByTagName("bidder");
         unsigned int bidCount=bidders.getLength();
         for(unsigned int j=0;j<bidCount;j++)
         {
              DOM_Node bidElem=bidders.item(j);
              DOMString person_ref=((DOM_Element &)bidElem).
              getElementsByTagName("personref").item(0).getAttributes().item(0).getNodeValue();
               for(unsigned int k=0;k<persCount;k++)
               {
                  DOM_Node personElem=pers.item(k);
                  if(personElem.getNodeType()==1)
                   {      DOMString person_id=personElem.getAttributes().item(0).getNodeValue();
                         if((person_ref.equals(person_id))&&(mark[k]==false)){
                             name[m++]=(DOM_Element &)(((DOM_Element &)personElem).
                                        getElementsByTagName("name").item(0));
                          mark[k]=true;}
                          }
```

```
            }
          }
        }
    }}}
```

In this example, we can observe that queries should traverse a collection of nodes on different two sub-trees. The second solution has to search for each bidder's *id* from the sub-tree rooted at node *open_auctions* and look up the corresponding name associated with the *id* in another one rooted at node *people*. Thus, applications need to resort to nested loop structure which makes the program complex. The second one can get a collection of bidder's id and a collection of person's id straightforwardly. The rest is to invoke a method *intersect()* to return the expected person's id, and then get their corresponding names of persons directly.

Q8 Get the names of persons who bid on items from Africa. (join person, open_auctions, item)

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_Element   siteElem=doc.getDocumentElement();
DOM_NodeList   personsid= siteElem.getElementbyTagName(“people”).
                              getChildren(1).getAttributeList().getChildren(0);
DOM_NodeList openNodes=siteElem. getElementbyTagName(“open_auctions”).getChildren(1);
DOM_NodeList itemrefNodes=openNodes. getChildren(1).filterTagName("itemref").
                              getAttributeList().getChildren(0);
DOM_NodeList regions_africa=siteElem.getElementbyTagName(“regions”). getChildren(1).
                        filterTagName("africa").getChildren(1).getAttributeList().getChildren(0);
DOM_NodeList itemsinAfrica=itemrefNodes.intersect (regions_africa);
DOM_NodeList biddersid=itemsinAfrica.getParents().getParents().getParents().getParents().
                              getChildren(1).filterTagName("bidder").getChildren(1).
                              filterTagName("personref").getAttributeList().getChildren(0);
DOM_NodeList bidder_person=personsid.intersect(biddersid,true);
DOM_NodeList biddersName=bidder_person.getParents().getParents().
                              getChildren(1).filterTagName(“name”);
```

- Solution using the DOM interface

```
DOM_Document doc=parser.getDocument();
DOM_Element   siteElem=doc.getDocumentElement();
DOM_NodeList  opens=siteElem.getElementbyTagName(“open_auctions”).
                              item(0).getChildNodes();
unsigned int opensCount=opens.getLength();
DOM_NodeList items= ((DOM_Element &)(siteElem.getElementsByTagName("regions").item(0))).
                           getElementsByTagName("africa").item(0).getChildNodes();
unsigned int itemCount=items.getLength();
DOM_NodeList   persons=siteElem.getElementbyTagName(“people”).
                               item(0).getChildNodes();
unsigned int persCount=persons.getLength();
DOM_Element person_name[MAXLENGTH];
bool mark1[MAXLENGTH],mark2[MAXLENGTH];

for(unsigned int n=0;n<MAXLENGTH;n++){    mark2[n]=false;
                                          mark1[n]=false;}
for (unsigned int i=0;i<opensCount;i++)
 {  DOM_Node openElem=opens.item(i);
    if(openElem.getNodeType()==1)
    {
      DOM_String item_id= ((DOM_Element &)openElem).getElementbyTagName(“itemref”).
                           item(0).getAttributes().item(0).getNodeValue();
     for(unsigned int j=0;j<itemCount;j++)
     {
      DOM_Node itemElem=items.item(j);
      if(items.getNodeType()==1)
      {DOMString item_africa=itemElem.getAttributes().item(0).getNodeValue();
       if((item_id.equals(item_africa))&&(mark1[j]==false)){
       mark1[j]=true;
       DOM_NodeList bidders=((DOM_Element &)openElem).getElementsByTagName("bidder");
```

```
 unsigned int bidderCount=bidders.getLength();
for(unsigned int l=0;l<bidderCount;l++)
{
 DOMString bidder_id=((DOM_Element &)(bidders.item(l))).
 getElementsByTagName("personref").item(0).getAttributes().item(0).getNodeValue();
  for(unsigned int k=0;k<persCount;k++)
    {
     DOM_Node personElem=persons.item(k);
     if(personElem.getNodeType()==1)
      {
        if((personElem.getAttributes().item(0).getNodeValue().equals(bidder_id))
         &&(mark2[k]==false))
           { mark2[k]=true;
             person_name[m++]=(DOM_Element &)(((DOM_Element &)personElem).
                                    getElementsByTagName("name").item(0));
             break;}
      }
     }
   }
  }}
 }}
```

In this example, queries based on the original DOM needs to traverse a collection of nodes in three different sub-trees with more complex loops. But the solution based on the extension retains relative simplicity.

5.3.5 Difference

Q9 List all ids of persons who are only sellers and do not buy other items in closed auctions.

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_Element   site=doc.getDocumentElement();
DOM_NodeList   closed_auction=site.
```

```
                    getElementbyTagName(“closed_auctions”).getChildren(1).getChildren(1);
DOM_NodeList  sellers=closed_auction.filterTagName(“seller”).getAttributeList().getChildren(0);
DOM_NodeList  buyers=closed_auction.filterTagName(“buyer”). getAttributeList().getChildren(0);
DOM_NodeList  dis_seller=sellers.subtract(buyers,true);
```

- Solution using the DOM interface

```
DOM_Document doc=parser.getDocument();
DOM_NodeList   closed_auctions=doc.getDocumentElement().
                    getElementbyTagName(“closed_auctions”).item(0).getChildNodes();
unsigned int closedCount= closed_auctions.getLength();
DOM_Node sellerAttr, seller_id[MAXLENGTH],id[MAXLENGTH];
bool mark[MAXLENGTH];

for(i=0;i<closedCount;i++)
{
 DOM_Node closedElem=closed_auctions.item(i);
 if(closedElem.getNodeType()==1)
 {DOM_Node seller=((DOM_Element &)closedElem).getElementsByTagName("seller").item(0);
   for(j=0;j<closedCount;j++)
   {
     DOM_Node closedNode=closed_auctions.item(j);
     DOM_Node buyer=((DOM_Element &)(closedNode)).getElementsByTagName("buyer").item(0);
     sellerAttr=seller.getAttributes().item(0);
     DOMString seller_id=sellerAttr.getNodeValue();
     DOMString buyer_id=buyer.getAttributes().item(0).getNodeValue();
     if(seller_id.equals(buyer_id))          break;
    }
 }
  if(j==closedCount) seller_id[k++]=sellerAttr; }
 }
}
for(i=0;i<k;i++)  mark[i]=false;
```

```
for(i=0;i<k;i++)
{
 if(mark[i]==false)
 {
   id[n++]=seller_id[i];
   for(j=i+1;j<k;j++)
     if(seller_id[i].getNodeValue().equals(seller_id[j].getNodeValue())) mark[j]=true;
 }
}
```

This is an application of self-join on node sets in one sub-tree rooted at *closed_auctions*. The second solution has to use complex comparison algorithm. The first solution is a simple implementation in terms of coding.

5.3.6 Sort

Q10 List in document order those open auctions on which all buyers watch. (join closed_auctions, people, open_auctions)

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_Element   siteElem=doc.getDocumentElement();
DOM_NodeList   personsid= siteElem.getElementbyTagName(“people”).
                              getChildren(1).getAttributeList().getChildren(0);
DOM_NodeList buyers=siteElem.getElementbyTagName(“closed_auctions”).
                          getChildren(1).getChildren(1). filterTagName(“buyer”).
                          getAttributeList().getChildren(0).distinct(true);
DOM_NodeList  buyer_persons=personsid.intersect(buyers);
DOM_NodeList   opens_id= buyer_persons.getParents().getParents().filterTagName(“watches”).
                              getChildren(1).getAttributeList().getChildren(0);
DOM_NodeList  open_auction=siteElem. getElementbyTagName(“open_auctions”).
                                   getChildren(1).getAttributeList().getChildren(0);
DOM_NodeList    result_auctions=open_auction.intersect(opens_id).
                    getParents().getParents().sort(TRUE);
```

- Solution using the DOM interface

```
DOM_Document doc=parser.getDocument();
DOM_Element   siteElem=doc.getDocumentElement();
DOM_NodeList   persons= siteElem.getElementsByTagName(“people”).item(0).
                                getElementsByTagName(“person”);
unsigned int personCount = persons.getLength();
DOM_NodeList open_auctions= siteElem.getElementbyTagName(“open_auctions”).
                                    item(0).getChildNodes();
unsigned int openCount = open_auctions.getLength();
DOM_NodeList closed_auctions= siteElem.getElementbyTagName(“closed_auctions”).
                                    item(0).getChildNodes();
unsigned int closedCount = closed_auctions.getLength();


for( i=0;i<openCount;i++)
{
 DOM_Node openElem=open_auctions.item(i);
 if(openElem.getNodeType()==1)
 {mark[i]=false;
  DOMString open_id=openElem.getAttributes().item(0).getNodeValue();
  for(j=0;j<closedCount;j++)
  {
    DOM_Node closedElem=closed_auctions.item(j);
    if(closedElem.getNodeType()==1)
    {DOM_Node buyer=((DOM_Element &)(closedElem)).
                                getElementsByTagName("buyer").item(0);
     DOMString buyer_id=buyer.getAttributes().item(0).getNodeValue();
     for(k=0;k<personCount;k++)
     {
      DOM_Node personElem=persons.item(k);
      DOMString person_id=personElem.getAttributes().item(0).getNodeValue();
      if(buyer_id.equals(person_id))   {
        DOM_NodeList watchesList=((DOM_Element &)personElem).
```

```
                                        getElementsByTagName("watches");
          if(watchesList.getLength()>0){
            DOM_NodeList watches=watchesList.item(0).getChildNodes();
            unsigned int watchesCount=watches.getLength();
           for(m=0;m<watchesCount;m++)
           {
             DOM_Node watchElem=watches.item(m);
             if(watchElem.getNodeType()==1)
             {
               DOMString watch_id=watchElem.getAttributes().item(0).getNodeValue();
                if((open_id.equals(watch_id))&&(mark[i]==false))
                {
                   openAuct[n++]=openElem;
                   mark[i]=true;
                   break;
                }
             }
           }
        }
     }
     }
     }
     }
}}
```

To get the expected *open_auctions* nodes in document order, the second solution needs to consider the sequence of loops. If the loop of nodes *open_auction* is placed in the outer, the ordered result can be obtained. However, the extended solution can obtained the ordered *open_auction* nodes very conveniently by invoking one method sort().

5.3.7 Constructing complicated results

Q11 Get the first five and the last five closed_auctions in document order, for which the seller does

not buy other items in closed auctions, return seller, buyer, item and price.

- Solution using the extension to the DOM interfaces.

```
DOM_Document doc=parser.getDocument();
DOM_NodeList   siteChilds=doc.getDocumentElement().createNodeList(1).getChildren(1);
DOM_NodeList  closed_auction=siteChilds.filterTagName("closed_auctions").
                                        getChildren(1).getChildren(1);
DOM_NodeList  sellers=
            closed_auction.filterTagName(“seller”). getAttributeList().getChildren(0);
DOM_NodeList  buyers=
            closed_auction.filterTagName(“buyer”). getAttributeList().getChildren(0);
DOM_NodeList  final_sellers=sellers.substract(buyers). getParents().getParents();
DOM_NodeList  closed_auctions=final_sellers.getParents();
DOM_NodeList  closed_sort=closed_auctions.sort(false);
unsigned int sortlen=closed_sort.getLength();
DOM_NodeList final_auctions;
if(sortlen<5)
     final_auctions=closed_sort.subList(0,sortlen);
else
{
    DOM_NodeList final1_auctions=closed_sort.subList(0,5);
    DOM_NodeList final2_auctions;
    if((sortlen>5)&&(sortlen<10))
      final2_auctions=closed_sort.subList(sortlen-6,sortlen-5);
    else
      final2_auctions=closed_sort.subList(sortlen-6,5);
    final_auctions=final1_auctions.catenate(final2_auctions);
 }
 DOM_NodeList finals=final_auctions.getChildren(1);
 final_sellers=finals.filterTagName("seller");
 DOM_NodeList  final_prices=finals. filterTagName(“price”);
 DOM_NodeList  final_buyers=finals.filterTagName(“buyer”);
 DOM_NodeList  final_items=finals. filterTagName(“itemref”);
```

- Solution using the DOM interface

```
DOM_Document doc=parser.getDocument();
DOM_Element siteElem=doc.getDocumentElement();
DOM_NodeList closed_auctions=siteElem.getElementsByTagName("closed_auctions").
                                        item(0).getChildNodes();
unsigned int closedCount = closed_auctions.getLength();
DOM_Node closedElem,sellerElem,closedNode,buyerElem;
DOM_Node seller[MAXLENGTH],buyer[MAXLENGTH],
                item[MAXLENGTH],price[MAXLENGTH];
DOM_Node selRes[20],buyRes[20],itemRes[20],pricRes[20];int k=0;
for( i=0;i<closedCount;i++)
{
 closedElem=closed_auctions.item(i);
 if(closedElem.getNodeType()==1)
 {
  sellerElem=((DOM_Element &)closedElem).getElementsByTagName("seller").item(0);
  for(j=0;j<closedCount;j++)
  {
    closedNode=closed_auctions.item(j);
    buyerElem=((DOM_Element &)(closedNode)).getElementsByTagName("buyer").item(0);
    DOMString seller_id=sellerElem.getAttributes().item(0).getNodeValue();
    DOMString buyer_id=buyerElem.getAttributes().item(0).getNodeValue();
     if(seller_id.equals(buyer_id))   break;
    }
  }
 }
 if(j==closedCount) {
      seller[k]=sellerElem;
      buyer[k]=((DOM_Element &)(closedElem)).getElementsByTagName("buyer").item(0);
      item[k]=((DOM_Element &)(closedElem)).getElementsByTagName("itemref").item(0);
      price[k]=((DOM_Element &)(closedElem)).getElementsByTagName("price").item(0);
```

```
        k++; }
            }

if(k<5) n=k; else n=5;
for(i=0;i<n;i++)
{     selRes[i]=seller[i];
       buyRes[i]=buyer[i];
       itemRes[i]=item[i];
       pricRes[i]=price[i];
}
if(k>=5){
 if((k>5)&&(k<10)) n=k-5; else n=10;
for(i=0;i<n;i++)
 {     selRes[2+i]=seller[k-1-i];
       buyRes[2+i]=buyer[k-1-i];
       itemRes[2+i]=item[k-1-i];
       pricRes[2+i]=price[k-1-i];
 }
}
```

For this query with more complex relations among sub-trees and sorting, the solution based on the original DOM requires significantly more code than the one based on the extension.

5.4 Experiments and Anaysis

Several factors, such as running speed, API simplicity, and storage usage, are often used to evaluate the performance of systems. Here, we do not consider the storage, since our extensions to DOM do not significantly affect the amount of space needed for the DOM structure themselves. Furthermore, with the falling price of RAM and the rising demand of high performance system, DOM meets the target of small real-time systems which do not process huge amounts of data. Real-time systems often run in distributed environment, thus the running time of the entire system includes processing time of the server and client and communication overhead between client and server.

We implemented the extended DOM and then conducted experiments to test the processing time based on the examples above in a single machine environment. The environment that runs the examples is a PC with 512MB of main memory, a 10GB hard disk and Pentium CPU clocked at 2.4GHz. The installed operating system is windows2000. For simplicity, we did not use CORBA to communicate between server code and client code, but rather implemented all code within one process. The data generator to create XML documents is XMark. We use different sizes of documents from 100KB to 50MB to evaluate performance (Table 5.1 lists different data size of XML document generated by the data generator). The implementation is based on Xerces, as explained in Section 4.5.

XML data(B) / the number of	116K	211K	460K	906K	2.45M	5.7M	11.4M	23.5M	46.5M
open_ auction nodes	12	24	48	96	240	600	1200	2400	4800
closed_auction nodes	10	19	38	77	195	489	975	1950	3900
item nodes	22	43	86	173	435	1087	2175	4350	8700
person nodes	25	51	102	204	510	1275	2550	5100	10200
bidder nodes	60	81	196	439	1140	2989	6182	12097	23521

Table 5.1 Different sizes of XML data

The preprocessing time includes parsing the document and converting it to a tree structure in the main memory. The parsing time with different sizes of data is shown as Figure 5.4 (but note the logarithmic scale on the X-axis). Because we make few changes to variables of the classes which be used in the parsing procedure, the parsing time of the extension has no changes from the original DOM.

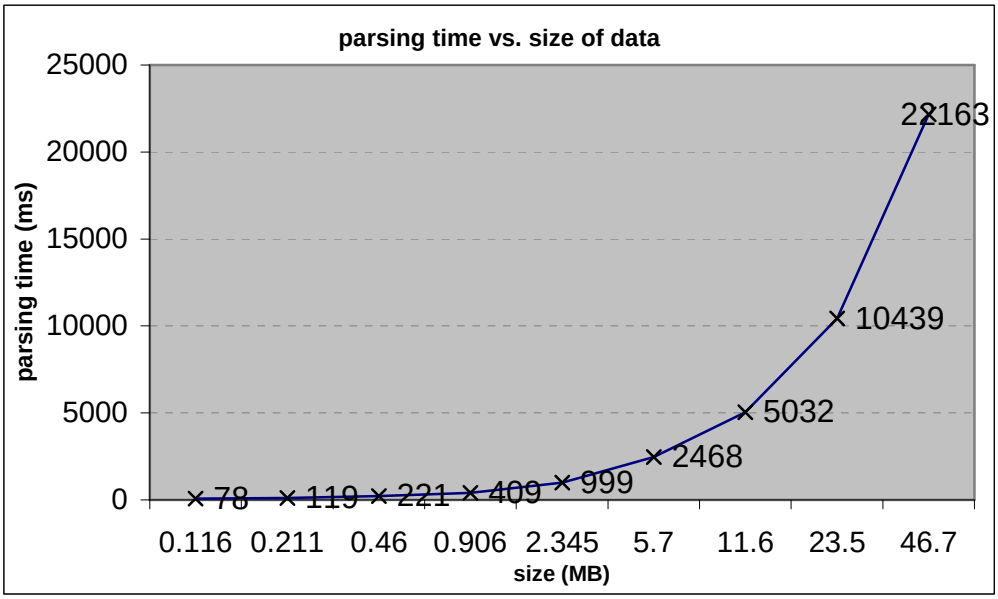


Figure 5.4 The parsing time with different size of data

Table 5.2 shows the processing time of our examples with different data size (the comparson graph of each example based on the DOM and the extension with different data sizes is shown in Appendix E). The time includes the parsing in the server and in the client as well as interprocess communication. As DOM features main-memory processing and both the server and client are on the same processors, the operations are quite fast when the size of data is very small. For both the application based on the DOM and the one based on the extension, the processing time is less than 1ms when the size of data is less than 900KB. We also find that the processing time of the first four examples for the extension is relatively slower compared with the DOM application. but with the increase of the size of data, the difference of processing time will decrease. Furthermore, after using the extension, the processing time of the last seven examples is less than the one based on the DOM, and the difference of the processing time based on the DOM and the extension increases with the size of XML data and complexity of query. The reason is that implementations based on the DOM take more time processing the nested loop structure than those based on the extension. For example 10, we do not give the processing time based on the DOM when the size of data is large than 2.45MB because of the unacceptable processing time.

		906KB	2.345MB	5.7MB	11.6MB	23.5MB	46.7MB
Q1	using the DOM	0	31ms	62ms	125ms	282ms	735ms
	using the extension	0	31ms	76ms	149ms	309ms	657ms
Q2	using the DOM	0	15ms	31ms	62ms	141ms	297ms
	using the extension	0	24ms	57ms	109ms	225ms	469ms
Q3	using the DOM	0	16ms	31ms	94ms	171ms	375ms
	using the extension	0	16ms	39ms	83ms	167ms	330ms
Q4	using the DOM	0	8ms	15ms	32ms	78ms	203ms
	using the extension	0	16ms	57ms	109ms	219ms	469ms
Q5	using the DOM	16ms	31ms	71ms	172ms	375ms	1000ms
	using the extension	0	28ms	68ms	141ms	281ms	578ms
Q6	using the DOM	15ms	16ms	63ms	172ms	375ms	968ms
	using the extension	0	24ms	73ms	141ms	281ms	580ms
Q7	using the DOM	32ms	266ms	2672ms	10.78s	39.6s	164s
	using the extension	0	31ms	203ms	1s	6.094s	28.3s
Q8	using the DOM	16ms	47ms	203ms	657ms	2735ms	10.4s
	using the extension	0	63ms	157ms	367ms	930ms	2.77s
Q9	using the DOM	32ms	250ms	1672ms	7984ms	38.7s	225s
	using the extension	0	31ms	78ms	281ms	1.418s	6.19s
Q10	using the DOM	2.44s	150s	_____	_____	_____	____
	using the extension	31ms	109ms	500ms	1781ms	8.328s	39.7s
Q11	using the DOM	31ms	250ms	1750ms	7875ms	38.5s	205s
	using the extension	16ms	46ms	218ms	781ms	3.328s	14s

Table 5.2 The processing time with different size of data

In a distributed environment, communication overhead impacts the application performance because of latency inherent in the network and bandwidth of the system. Fewer message exchanges between client and server reduce total remote access delays and thus improve the system throughput. We compare the number of messages required by an application of the DOM to the number required for our extension. In Table 5.3 we provide a comparative analysis, where we let n_1 be the number of *items*, n_2 be the number of *open_auction*, n_3 be the number of *person*, n_4 be the number of *closed auction*, n_5 be the number of *bidders*, and the notation (a, b) denote the bound range of messages.

	The solution based on DOM	The solution based on the extension
Example 1	$7n_1+80$	9
Example 2	$(13n_2+8, 14n_2+8)$	11
Example 3	$6n_2+8$	5
Example 4	$6n_1+80$	7
Example 5	$7n_5+6n_2+8$	11
Example 6	$(10+6n_2, 7n_5+3n_2+6)$	13
Example 7	$(12+6n_2,14n_5n_3+22n_2+14n_5+12)$	28
Example 8	$O(n_2\ n_1\ n_3)$	34
Example 9	$(17\ n_4+8, 11n_4^2+6\ n_4+8)$	12
Example 10	$O(n_2\ n_4\ n_3)$	28
Example 11	$(17n_4+8,10n_4^2+12n_4+8)$	(25,28)

Table 5.3 Communication messages between client and server

Consider the communication messages based on the DOM. The first four examples based on the DOM are relatively simple to analyze. Example 1 and 4 contain nested loops, where the outer loop iterates on six *regions*, but seven extra iterations are also needed to process white space. They also scan each *item* node in the inner for loop. Therefore, the communication messages are direct proportional to the number of items. From Figure 5.5, when updating on a 5.7MB XML database which has 1087 items, 7689 communication messages are required. With the increase of data, the communication messages increase greatly. Similarly, in the Example 2 and 3, the amount of communication messages is directly proportional to the number of open auctions. Firgure 5.6 shows the number of communication messages with the size of *open auction* in the example 3.

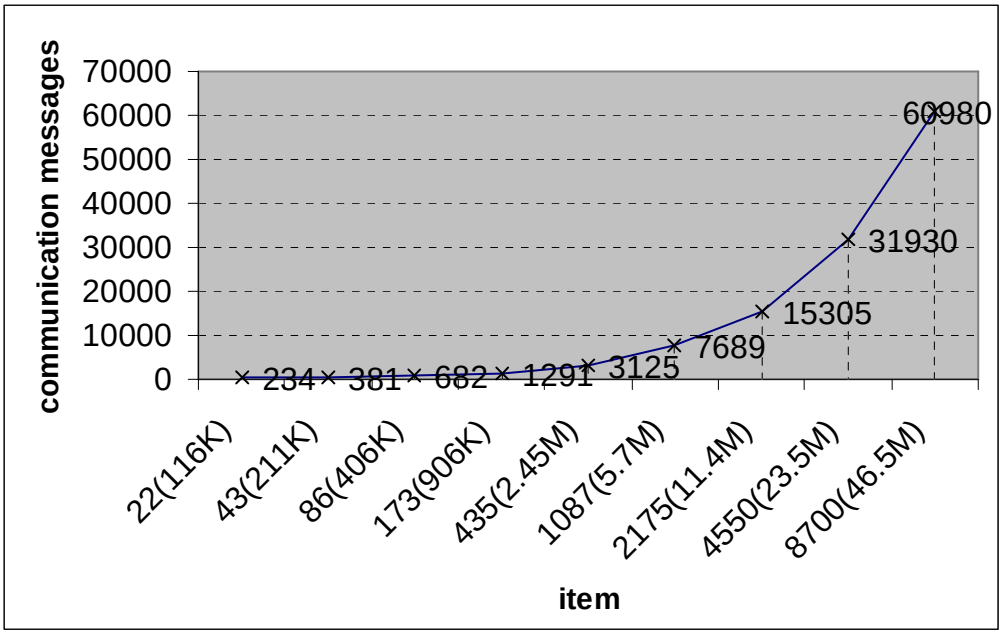


Figure 5.5 The commnunication messages with different data in Example 1

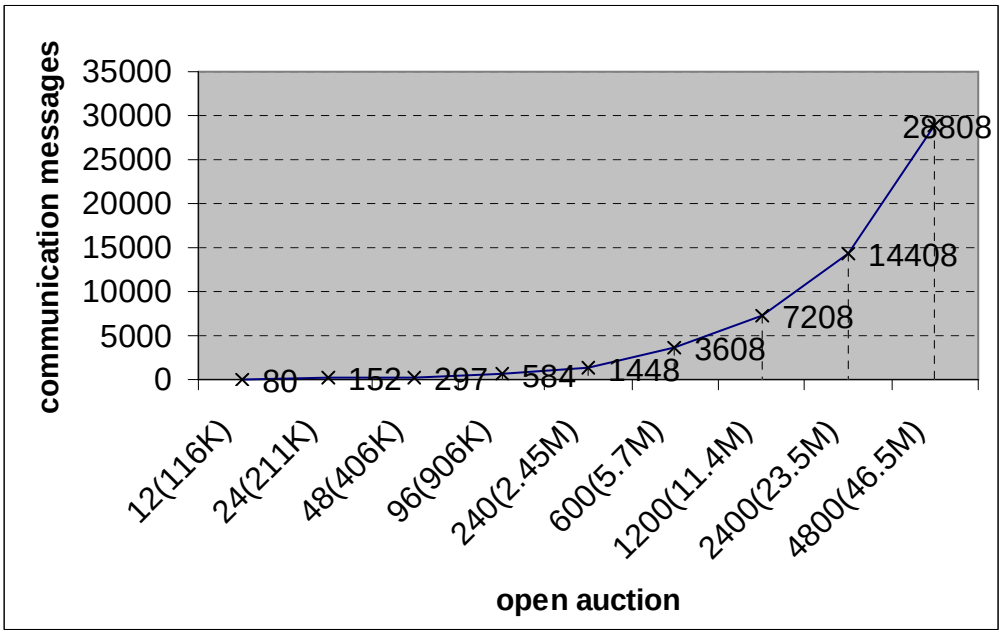


Figure 5.6 The commnunication messages with different data in Example 3

Example 5 and Example 6 have a little more complexity than the first four examples, since the number of messages varies with the size of *open_auction* and *item*. For example, when there is only

460KB, 1668 commnunication messages are required in Example 5, which is greater than what is needed in the first four examples with 906KB data volume. The reasonable explanation is that they need to scan more nodes than those in the first four examples with the same size of XML data. The last five examples have much more complex relationship among nodes and need to join on different sub-trees, applications need to use more loops to handle large amount of nodes. Consequently, a large number of messages have to be conveyed back and forth due to the complexity of query.

When using DOM, the communication messages between client and server vary with the amount of XML data and the complexity of queries. Since the original DOM provides operations based on one node, when applications need to get node sets with repeated features, they have to use loops to process each of these nodes. This is made worse when we parse XML data with no DTD, since extra iterations are needed for processing white space. Therefore, a large amount of communication messages are needed back and forth between client and server, and the processing time will slow down greatly when the data volume is large. On the contrary, since our extension provides operations on node sets, applications get collections of nodes in one operation directly. Thus, communication messages will be reduced sharply. Interestingly, for all of our sample queries, the number of communication messages is independent of the data volume.

Thus with our extension, the number of communication messages between client and server will greatly reduced in the server and in the client is also reduced for large date size. We conclude that when the amount of XML data is large, the running time of the entire system using the extension will be reduced greatly. This benefit is received at the same time that our extended DOM simplifies the programming coding greatly and thus is more convenient and less error-prone for application programmers.

Chapter 6
Conclusions and future work

XML documents are modeled as a virtual tree structure by DOM. Applications access and update the content and structure of documents dynamically through this platform-neutral and language-neutral interface. Such applications visit objects of the trees through the Node interface and use loops to visit all children of a node. We propose some extensions to the DOM NodeList interface to manipulate collections of nodes set-at-a-time. This provides a more powerful API that makes XML document retrieval faster and more convenient.

The contributions of this research can be summarized from two aspects. First, we propose extensions to the DOM interface that can directly traverse the DOM tree set-at-a-time, explain its importance, and give descriptions using IDL and C++ binding for it. The idea to support operations for sets of nodes comes from other environments in which set-at-a-time data access has been shown to be worthwhile. We start by extending the NodeList interface with two types of operations: those that map Node operations to NodeLists and those that manipulate the NodeList itself. These operations borrowed from different software language sources: XQuery and XPath operators, APL and MAPLE.

Second, we implemented the extensions to validate our ideas and demonstrate its efficiency through a test system. We give examples by using the DOM and our extended DOM, comparing their performance. The result of the test shows it is useful to simplify the applications programming and reduce the communication messages between client and server after the extensions.

The extended DOM inherits all the advantages of the original DOM. Since our extended DOM only adds some new methods, it does not change any methods in DOM, and thus all of the methods in the DOM interfaces can still be used. In addition, the extensions fit the same navigational philosophy of DOM, thus it is natural to use them with DOM.

Finally, the extended DOM results in significant benefits to applications. It can cope with collections of nodes set-at-a-time efficiently. More specifically, application code can be simplified greatly and

communication messages between client and server will be decreased significantly.

We expect our research work to evolve in the future. In the short term, we need to add some more functions to the NodeList interface to provide applications with more abilities.

- Our extensions are not very efficient when dealing with sorting. In our *sort* function, we only reorder node sets by document order. But in many cases applications need to order elements by value especially when text nodes in the node sets contain numeric values. For example, an application may want to query the XML document named “auction.xml” to gain the initial increases of all open auctions. In addition, we want to return a specific type of nodes. Consequently, we should modify the method *sort(bool order)* to give more functionality.
- When comparing two collections of node sets, we provide three methods, *distinct(bool byValue)*, *subtract(bool byValue)* and *intersect(bool byValue)*. The parameter *byValue* is only applicable for text nodes. We need to extend it to fit all types of nodes and only return the expected type of nodes.

Therefore, some methods that are challenging to implement will be

- *sort(bool order, unsigned short nodetype, bool isValue);*
  Resort node sets whose type are nodetype, other type of nodes are discarded. If the nodetype is text node and *isValue* is true, then sort them by node value increasingly or decreasingly.
- *distinct(bool byValue,unsigned short nodetype);*
  Eliminate duplicates by value or object and only those nodes with specific node type *nodetype* are returned.
- *subtract(NodeList list, bool byValue,unsigned short nodetype);*
  Compare two node sets by value or object, and return the difference of the two node sets whose node type are *nodetype*.
- *intersect(NodeList list, bool byValue,unsigned short nodetype);*
  Compare two node sets by value or object, and return the difference of the two node sets whose node type are *nodetype*.

In addition, we are reluctant to suggest superabundant extensions for the convenience to applications, and we need to balance between processing speed and communication overhead. Thus, how to evaluate this balance need further research.

Finally, communication cost based on a practical system in the distributed environment remains to be tested, and a quantitative analysis of the relationship between communication messages and throughput needs to be evaluated.

In the long term, we plan to build an economic DBMS to manage XML data for small businesses. This will require that the implementation be extended to manage data on secondary storage. It is hoped that the resulting system would find applications in areas that require high performance XML processing without the full power of XQuery or similar high-level languages.

Appendix A

“item.xml”

Sample data of “item.xml”

```
 <regions>
  <africa>
   <item id=“item0”>
     <name>duteous nine eighteen</name>
     <location>United States</location>
     <quantity>1</quantity>
     <shipping>Will ship internationally, See description for charges</shipping>
     <incategory category=“category2”/>
   </item>
   <item id=“item1”>
     <name>iron</name>
     <location>China</location>
     <quantity>101</quantity>
     <shipping>Will ship internationally</shipping>
     <incategory category=“category2”/>
   </item>
  </africa>
  <europe>
   <item id=“item2”>
     <name>paper</name>
     <location>United States</location>
     <quantity>10</quantity>
     <shipping>Will ship internationally </shipping>
     <incategory category=“category3”/>
   </item>
   <item id=“item3”>
     <name>steel</name>
     <location>France</location>
     <quantity>1000</quantity>
     <shipping>Will ship internationally </shipping>
     <incategory category=“category4”/>
     <incategory category=“category5”/>
   </item>
  </europe>
 </region>
```

Appendix B

"price.xml"

The data is also available in Data for Q10 of [CFM+01].

```
<prices>
   <book>
      <title> Advanced Programming in the Unix environment</title>
      <source>www.amazon.com</source>
      <price>65.95</price>
   </book>
   <book>
      <title>Advanced Programming in the Unix envornment</title>
      <source>www.bn.com</source>
      <price>65.95</price>
   </book>
   <book>
     <title>TCP/IP Illustrated </title>
     <source>www.amazon.com</source>
     <price>65.95</price>
   </book>
   <book>
     <title>TCP/IP Illustrated</title>
     <source>www.bn.com</source>
     <price>65.95</price>
   </book>
   <book>
     <title>Data on the Web</title>
     <source>www.amazon.com</source>
     <price>34.95</price>
   </book>
   <book>
    <title>Data on the Web</title>
    <source>www.bn.com</source>
    <price>39.95</price>
   </book>
</price>
```

Appendix C

“auction.dtd”

The file named “auction.dtd” is also available as http://monetdb.cwi.nl/xml/index.html.

```
<!-- DTD for auction database -->
<!-- $Id: auction.dtd,v 1.15 2001/01/29 21:42:35 albrecht Exp $ -->
<!ELEMENT site(regions, categories, catgraph, people, open_auctions, closed_auctions)>
<!ELEMENT categories(category+)>
<!ELEMENT category(name, description)>
<!ATTLIST category id ID #REQUIRED>
<!ELEMENT name(#PCDATA)>
<!ELEMENT description(text | parlist)>
<!ELEMENT text(#PCDATA | bold | keyword | emph)*>
<!ELEMENT bold(#PCDATA | bold | keyword | emph)*>
<!ELEMENT keyword(#PCDATA | bold | keyword | emph)*>
<!ELEMENT emph(#PCDATA | bold | keyword | emph)*>
<!ELEMENT parlist(listitem)*>
<!ELEMENT listitem(text | parlist)*>
<!ELEMENT catgraph(edge*)>
<!ELEMENT edge EMPTY>
<!ATTLIST edge from IDREF #REQUIRED to IDREF #REQUIRED>
<!ELEMENT regions(africa, asia, australia, europe, namerica, samerica)>
<!ELEMENT africa(item*)>
<!ELEMENT asia(item*)>
<!ELEMENT australia(item*)>
<!ELEMENT namerica(item*)>
<!ELEMENT samerica(item*)>
<!ELEMENT europe(item*)>
<!ELEMENT item(location,quantity,name,payment,description,shipping,incategory+,mailbox)>
<!ATTLIST item id ID #REQUIRED featured CDATA #IMPLIED>
<!ELEMENT location(#PCDATA)>
<!ELEMENT quantity(#PCDATA)>
<!ELEMENT payment(#PCDATA)>
<!ELEMENT shipping(#PCDATA)>
<!ELEMENT reserve(#PCDATA)>
<!ELEMENT incategory EMPTY>
<!ATTLIST incategory category IDREF #REQUIRED>
<!ELEMENT mailbox(mail*)>
<!ELEMENT mail(from, to, date, text)>
<!ELEMENT from(#PCDATA)>
<!ELEMENT to(#PCDATA)>
<!ELEMENT date(#PCDATA)>
<!ELEMENT itemref EMPTY>
<!ATTLIST itemref item IDREF #REQUIRED>
<!ELEMENT personref EMPTY>
```

```
<!ATTLIST personref person IDREF #REQUIRED>
<!ELEMENT people(person*)>
<!ELEMENT person(name,emailaddress,phone?,address?,homepage?,creditcard?,profile?,
                 watches?)>
<!ATTLIST person id ID #REQUIRED>
<!ELEMENT emailaddress(#PCDATA)>
<!ELEMENT phone(#PCDATA)>
<!ELEMENT address(street, city, country, province?, zipcode)>
<!ELEMENT street(#PCDATA)>
<!ELEMENT city(#PCDATA)>
<!ELEMENT province(#PCDATA)>
<!ELEMENT zipcode(#PCDATA)>
<!ELEMENT country(#PCDATA)>
<!ELEMENT homepage(#PCDATA)>
<!ELEMENT creditcard(#PCDATA)>
<!ELEMENT profile(interest*, education?, gender?, business, age?)>
<!ATTLIST profile income CDATA #IMPLIED>
<!ELEMENT interest EMPTY>
<!ATTLIST interest category IDREF #REQUIRED>
<!ELEMENT education(#PCDATA)>
<!ELEMENT income(#PCDATA)>
<!ELEMENT gender(#PCDATA)>
<!ELEMENT business(#PCDATA)>
<!ELEMENT age(#PCDATA)>
<!ELEMENT watches(watch*)>
<!ELEMENT watch EMPTY>
<!ATTLIST watch open_auction IDREF #REQUIRED>
<!ELEMENT open_auctions(open_auction*)>
<!ELEMENT open_auction(initial, reserve?, bidder*, current, privacy?, itemref, seller, annotation,
                       quantity, type, interval)>
<!ATTLIST open_auction id ID #REQUIRED>
<!ELEMENT privacy(#PCDATA)>
<!ELEMENT initial(#PCDATA)>
<!ELEMENT bidder(date, time, personref, increase)>
<!ELEMENT seller EMPTY>
<!ATTLIST seller person IDREF #REQUIRED>
<!ELEMENT current(#PCDATA)>
<!ELEMENT increase(#PCDATA)>
<!ELEMENT type(#PCDATA)>
<!ELEMENT interval(start, end)>
<!ELEMENT start(#PCDATA)>
<!ELEMENT end(#PCDATA)>
<!ELEMENT time(#PCDATA)>
<!ELEMENT status(#PCDATA)>
<!ELEMENT amount(#PCDATA)>
<!ELEMENT closed_auctions(closed_auction*)>
<!ELEMENT closed_auction(seller, buyer, itemref, price, date, quantity, type, annotation?)>
<!ELEMENT buyer EMPTY>
```

```
<!ATTLIST buyer person IDREF #REQUIRED>
<!ELEMENT price(#PCDATA)>
<!ELEMENT annotation(author, description?, happiness)>
<!ELEMENT author EMPTY>
<!ATTLIST author person IDREF #REQUIRED>
<!ELEMENT happiness(#PCDATA)>
```

Appendix D

Some snippets of XML sample data

```
<site>
 <regions>
  <africa>
   <item id="item0">
    <location>United States</location>
    <quantity>1</quantity>
    <name>duteous nine eighteen</name>
    <description>
     <parlist>
      <listitem>
       <text>
          page reous lady authority capt professes stabs monster petition
          <keyword>officer embrace such distinction attires</keyword>
       </text>
      </listitem>……
     </parlist>
    </description>
    <shipping>Will ship internationally, See description for charges</shipping>
    <incategory category="category2"/>
    <incategory category="category1"/>
    <incategory category="category3"/>
    <incategory category="category3"/>
    <incategory category="category0"/>
    <mailbox>…..</mailbox>
   </item>….
  </africa>…
 </region>
 <categories>
   <category id="category0">
     …..
   </category>….
 </categries>
 <categraph>
    <edge from="category1" to="categrory1"/>
    …..
 </categraph>
 <people>
  <person id="person0">
     <name>Nobo Vijaykrishnan</name>
     <emailaddress>mailto:Vijaykrishnan@nyu.edu</emailaddress>
     <phone>+0(477)63141558</phone>
     <address>…</address>
     <homepage>http://www.nyu.edu/~Vijaykrishnan</homepage>
```

```
      <profile income="83146.23">
      <interest category="category1">
      <interest category="category1">
      <interest category="category2">
      <interest category="category3">
      <interest category="category1">
      <interest category="category2">
      <education>Graduate School</education>
      <gendor>male</gendor>
      <business>No</business>
      <age>52</age>
      </profile>
      <watches>
        <watch open_auction="open_auction34"/>
        <watch open_auction="open_auction49"/>
      </watches>
     </person>…
   </people>
   <open_auctions>
    <open_auction id="open_auction0">
      <initial>109.51</initial>
      <bidder>
        <date>05/26/2001</date>
        <time>16:06:53</time>
        <personref  person="person103"/>
        <increase>6.00</increase>
      </bidder>….
      <current>324.01</current>
      <itemref item="item0"/>
      <seller person="person64"/>
      <annotation>….</annotation>
      <quantity>2</quantity>
      <type>Featured</type>
      <interval>
        <start>04/19/2001</start>
        <end>07/11/2001</end>
      </interval>
    </open_auction>…..
</open_auctions>
<closed_auctions>
  <closed_auction>
    <seller person="person39">
    <buyer person="person58">
    <itemref item="item1">
    <price>13.18</price>
    <date>02/06/1999</date>
    <quantity>1</quantity>
    <type>Regular</type>
```

```
      <annotation>….</annotation>
    </closed_auction>….
  </closed_auctions>
</site>
```

Appendix E

Source code of examples

D:0 Source code of Head file

```
#include <sax/ErrorHandler.hpp>
#include <iostream.h>
#include <util/PlatformUtils.hpp>
#include <sax/SAXException.hpp>
#include <sax/SAXParseException.hpp>
#include <parsers/DOMParser.hpp>
#include <dom/DOM_DOMException.hpp>
#include <string.h>
#include <stdlib.h>
class SAXParseException;

// Simple error handler deriviative to install on parser
class DOMCountErrorHandler : public ErrorHandler
{
  public:
    DOMCountErrorHandler();
    ~DOMCountErrorHandler();

    bool getSawErrors() const;

    //  Implementation of the SAX ErrorHandler interface
    void warning(const SAXParseException& e);
    void error(const SAXParseException& e);
    void fatalError(const SAXParseException& e);
    void resetErrors();

    private :
      DOMCountErrorHandler(const DOMCountErrorHandler&);
      void operator=(const DOMCountErrorHandler&);

    //  Private data members. fSawErrors is set if we get any errors, and is queryable via a getter
   //method. It is used by the main code to suppress output if there are errors.
    bool    fSawErrors;
};

//  This is a simple class that lets us do easy (though not terribly efficient) trancoding of XMLCh data
//to local code page for display.
class StrX
{
   public :
    //  Constructors and Destructor
```

```
StrX(const XMLCh* const toTranscode)
{ // Call the private transcoding method
  fLocalForm = XMLString::transcode(toTranscode); }

   ~StrX()
   {
      delete [] fLocalForm;
   }

  const char* localForm() const
   {
      return fLocalForm;
   }

private :
   //  Private data members.  fLocalForm is the local code page form of the string.
   char*   fLocalForm;
};

DOMCountErrorHandler::DOMCountErrorHandler() :fSawErrors(false)
{
}

DOMCountErrorHandler::~DOMCountErrorHandler()
{
}

//  DOMCountHandlers: Overrides of the SAX ErrorHandler interface
void DOMCountErrorHandler::error(const SAXParseException& e)
{
   fSawErrors = true;
   cerr << "\nError at file " << StrX(e.getSystemId())<< ", line " << e.getLineNumber()
         << ", char " << e.getColumnNumber()<< "\n  Message: " << StrX(e.getMessage()) << endl;
}

void DOMCountErrorHandler::fatalError(const SAXParseException& e)
{
    fSawErrors = true;
    cerr << "\nFatal Error at file " << StrX(e.getSystemId())<< ", line " << e.getLineNumber()
        << ", char " << e.getColumnNumber()<< "\n  Message: " << StrX(e.getMessage()) << endl;
}

void DOMCountErrorHandler::warning(const SAXParseException& e)
{
     cerr << "\nWarning at file " << StrX(e.getSystemId())<< ", line " << e.getLineNumber()
         << ", char " << e.getColumnNumber()<< "\n  Message: " << StrX(e.getMessage()) << endl;
}
```

```
void DOMCountErrorHandler::resetErrors()
{
}

inline ostream& operator<<(ostream& target, const StrX& toDump)
{
   target << toDump.localForm();
   return target;
}

inline bool DOMCountErrorHandler::getSawErrors() const
{
   return fSawErrors;
}
```

D:1 Source code of Example1

- The extension solution:

```
int main()
{
 // Initialize the XML4C system
 try
   {
      XMLPlatformUtils::Initialize();
   }

   catch (const XMLException& toCatch)
   {
       cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
       return 1;
   }

   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                    doNamespaces    = false;
   bool                    doSchema        = false;

   xmlFile = "../../../../../samples/data/auction.xml";

   // Instantiate the DOM parser.
   DOMParser parser;
   parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);

   // create our error handler and install it
   DOMCountErrorHandler errorHandler;
   parser.setErrorHandler(&errorHandler);
```

```
//  Get the starting time and kick off the parse of the indicated file. Catch any exceptions that might
//  propogate out of it.
unsigned long duration;
try
{
    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
    parser.parse(xmlFile);
    const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
    duration = endMillis - startMillis;
}

catch (const XMLException& toCatch)
{
   cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
         << StrX(toCatch.getMessage()) << "\n" << endl;
   return -1;
}
catch (const DOM_DOMException& toCatch)
{
   cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
            << toCatch.code << "\n" << endl;
   XMLPlatformUtils::Terminate();
   return 4;
}
catch (...)
{
   cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
   XMLPlatformUtils::Terminate();
   return 4;
}

if (errorHandler.getSawErrors())
{
   cout << "\nErrors occured, no output available\n" << endl;
}
else
{  unsigned long staMil,endMil;
    cout<<xmlFile<<":"<<duration<<"ms"<<endl;
    staMil = XMLPlatformUtils::getCurrentMillis();
    DOM_Document doc = parser.getDocument();
    DOM_NodeList items=doc.getDocumentElement().getElementsByTagName("regions").
                                  getChildren(0).filterNodeType(1).getChildren(0).filterNodeType(1);
    unsigned int itemsCount = items.getLength();

     DOM_NodeList itemsChild=items.getChildren(0).filterNodeType(1);
     cout<<"items children length is "<<itemsChild.getLength()<<endl;
     DOM_NodeList description=itemsChild.filterTagName("description");
```

```
        cout<<"description="<<description.getLength()<<endl;
        items.removeChildren(description);
        itemsChild=items.getChildren(0).filterNodeType(1);
        cout<<"items children length is "<<itemsChild.getLength()<<endl;
        endMil = XMLPlatformUtils::getCurrentMillis();
        duration = endMil - staMil;
        cout<<"processing time:"<<duration<<endl;
      }

      // And call the termination method
      XMLPlatformUtils::Terminate();
      return 0;
   }
```

- The DOM solution:

```
//Delete all descriptions of items
int main()
{
   try
     XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   { cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
     return 1;
   }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
  bool                    doNamespaces    = false;
   bool                    doSchema        = false;
   xmlFile = "../../../../data/auction.xml";
   DOMParser parser;
   parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);

  DOMCountErrorHandler errorHandler;
  parser.setErrorHandler(&errorHandler);
  unsigned long duration;
  try
 {
   const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
   parser.parse(xmlFile);
   const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
   duration = endMillis - startMillis;
}
catch (const XMLException& toCatch)
{cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
        << StrX(toCatch.getMessage()) << "\n" << endl;
 return -1;
```

```
}
catch (const DOM_DOMException& toCatch)
{cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
        << toCatch.code << "\n" << endl;
  XMLPlatformUtils::Terminate();
 return 4;
}
catch (...)
{
   cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
    XMLPlatformUtils::Terminate();
    return 4;
}
if (errorHandler.getSawErrors())
   cout << "\nErrors occured, no output available\n" << endl;
else
{
   DOM_Document doc = parser.getDocument();
   DOM_NodeList regions=doc.getDocumentElement().getElementsByTagName("regions").
                                item(0).getChildNodes();
   // Print out the stats that we collected and time taken.
   cout << xmlFile << ": " << duration << " ms ("<< regions.getLength() << " elems)." << endl;
   for(unsigned int j=0;j<regions.getLength();j++)
   {
    DOM_NodeList items=regions.item(j).getChildNodes();
    for(unsigned int i=0;i<items.getLength();i++)
    {
    DOM_Node item_i=items.item(i);
    if(item_i.getNodeType()==1)
    {
     DOM_Node desc=((DOM_Element )item_i).getElementsByTagName("description").item(0);
     item_i.removeChild(desc);
    }}}}
XMLPlatformUtils::Terminate();
return 0;
}
```

D:2 Source code of Example2

- The extension solution

```
//close those open auctions whose initial price is 0
int main()
{
   try
     XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   {
      cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
```

```
  return 1;
}
const char*             xmlFile = 0;
DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
bool                    doNamespaces    = false;
bool                    doSchema        = false;

xmlFile = "../../../../../samples/data/auction.xml";
DOMParser parser;
parser.setValidationScheme(valScheme);
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);

unsigned long duration;
try
{
    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
    parser.parse(xmlFile);
    const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
    duration = endMillis - startMillis;
}
catch (const XMLException& toCatch)
{
   cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
        << StrX(toCatch.getMessage()) << "\n" << endl;
  return -1;
}
catch (const DOM_DOMException& toCatch)
{
    cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
         << toCatch.code << "\n" << endl;
    XMLPlatformUtils::Terminate();
    return 4;
}
catch (...)
{
   cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
    XMLPlatformUtils::Terminate();
    return 4;
}
if (errorHandler.getSawErrors())
    cout << "\nErrors occured, no output available\n" << endl;
 else
{
    unsigned long staMil,endMil;
    cout<<xmlFile<<":"<<duration<<"ms"<<endl;
```

```
      staMil = XMLPlatformUtils::getCurrentMillis();
      DOM_Document doc = parser.getDocument();
      DOM_NodeList   open_auctions=doc.getDocumentElement().
                          getElementsByTagName("open_auctions");
      unsigned int opensCount = open_auctions.getLength();
     DOM_NodeList initials=open_auctions.getChildren(0).getChildren(0).filterTagName("initial");
     DOM_NodeList iniVal=initials.getChildren(0).filterValue("0");
     DOM_NodeList openAuct=iniVal.getParents().getParents();
    open_auctions.subtractFromChildren(openAuct);
    endMil = XMLPlatformUtils::getCurrentMillis();
    duration = endMil - staMil;
    cout<<"processing time:"<<duration<<endl;
 }
```

- The DOM solution

```
//Delete all descriptions of items
int main()
{
 try
      XMLPlatformUtils::Initialize();

 catch (const XMLException& toCatch)
 {
   cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
   return 1;
 }

const char*             xmlFile = 0;
DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
bool                    doNamespaces    = false;
bool                    doSchema        = false;
xmlFile = "../../../../data/auction.xml";
DOMParser parser;
parser.setValidationScheme(valScheme);
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);
unsigned long duration;
try
{    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
     parser.parse(xmlFile);
     const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
     duration = endMillis - startMillis;
}
catch (const XMLException& toCatch)
{
```

```
    cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
         << StrX(toCatch.getMessage()) << "\n" << endl;
    return -1;
  }
  catch (const DOM_DOMException& toCatch)
  {
    cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
         << toCatch.code << "\n" << endl;
    XMLPlatformUtils::Terminate();
    return 4;
  }
  catch (...)
  {
    cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
    XMLPlatformUtils::Terminate();
    return 4;
  }

  if (errorHandler.getSawErrors())
    cout << "\nErrors occured, no output available\n" << endl;
   else
  {
    DOM_Document doc = parser.getDocument();
    DOM_NodeList regions=doc.getDocumentElement().getElementsByTagName("regions").
                                item(0).getChildNodes();
    cout << xmlFile << ": " << duration << " ms ("<< regions.getLength() << " elems)." << endl;
   for(unsigned int j=0;j<regions.getLength();j++)
   {
     DOM_NodeList items=regions.item(j).getChildNodes();
     for(unsigned int i=0;i<items.getLength();i++)
     {
       DOM_Node item_i=items.item(i);
       if(item_i.getNodeType()==1)
       {DOM_Node descript=((DOM_Element &)item_i).
                                getElementsByTagName("description").item(0);
         item_i.removeChild(descript);
       }
     }
} }
XMLPlatformUtils::Terminate();
return 0;
}
```

D:3 Source code of Example3

- The extension solution

```
//Get all intervals in all open auctions
int main()
```

```
{
   try
     XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   {
     cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
     return 1;
   }
 const char*             xmlFile = 0;
 DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
 bool                    doNamespaces    = false;
 bool                    doSchema        = false;
 xmlFile = "../../../../../samples/data/auction.xml";
 DOMParser parser;
 parser.setValidationScheme(valScheme);
 parser.setDoNamespaces(doNamespaces);
 parser.setDoSchema(doSchema);
 DOMCountErrorHandler errorHandler;
 parser.setErrorHandler(&errorHandler);
  unsigned long duration;
  try
  {    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
        parser.parse(xmlFile);
        const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
        duration = endMillis - startMillis;
   }
   catch (const XMLException& toCatch)
   {
     cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
             << StrX(toCatch.getMessage()) << "\n" << endl;
     return -1;
    }
    catch (const DOM_DOMException& toCatch)
    {
          cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
              << toCatch.code << "\n" << endl;
          XMLPlatformUtils::Terminate();
          return 4;
    }
    catch (...)
    {
         cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
          XMLPlatformUtils::Terminate();
          return 4;
    }
   if (errorHandler.getSawErrors())
         cout << "\nErrors occured, no output available\n" << endl;
   else
```

```
    {
       unsigned long staMil,endMil;
       cout<<xmlFile<<":"<<duration<<"ms"<<endl;
       staMil = XMLPlatformUtils::getCurrentMillis();
       DOM_Document doc = parser.getDocument();
       DOM_NodeList open_auction=doc.getDocumentElement().
                          getElementsByTagName("open_auctions").getChildren(1);
       unsigned int openCount = open_auction.getLength();
        DOM_NodeList interval=open_auction.getLastChildren(1);
       endMil = XMLPlatformUtils::getCurrentMillis();
       duration = endMil - staMil;
       cout<<"processing time:"<<duration<<endl;
    }
    XMLPlatformUtils::Terminate();
    return 0;
  }
```

- The DOM Solution

```
#define MAXLENGTH 1025
int main()
{
   try
     XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   {cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
    return 1;
   }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                  doNamespaces    = false;
   bool     doSchema        = false;
   xmlFile = "../../../../data/auction.xml";
  DOMParser parser;
   parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);
   DOMCountErrorHandler errorHandler;
   parser.setErrorHandler(&errorHandler);
   unsigned long duration;
   try
     {
         const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
         parser.parse(xmlFile);
         const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
         duration = endMillis - startMillis;
     }
```

```
    catch (const XMLException& toCatch)
    {
        cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
             << StrX(toCatch.getMessage()) << "\n" << endl;
        return -1;
    }
    catch (const DOM_DOMException& toCatch)
    {
        cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
             << toCatch.code << "\n" << endl;
        XMLPlatformUtils::Terminate();
        return 4;
    }
    catch (...)
    {
       cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
        XMLPlatformUtils::Terminate();
        return 4;
    }

    if (errorHandler.getSawErrors())
        cout << "\nErrors occured, no output available\n" << endl;
     else
    {
       unsigned int j=0;
        DOM_Element interval[MAXLENGTH];
        DOM_Document doc = parser.getDocument();
        DOM_NodeList open_auction=doc.getDocumentElement().
                              getElementsByTagName("open_auctions").item(0).getChildNodes();
        for(unsigned int i=0;i<open_auction.getLength();i++)
         {DOM_Node openElem=open_auction.item(i);
          if(openElem.getNodeType()==1)
          interval[j++]=(DOM_Element &)(((DOM_Element &)openElem).
                                        getElementsByTagName("interval").item(0));
         }
         cout<<"interval length is:"<<j<<endl;
    }
    XMLPlatformUtils::Terminate();
    return 0;
}
```

D:4 Source code of Example4

- The extension solution

```
//Get names of all items
int main()
{
   try
```

```
    XMLPlatformUtils::Initialize();

catch (const XMLException& toCatch)
{
     cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
     return 1;
}

const char*             xmlFile = 0;
DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
bool                    doNamespaces    = false;
bool                    doSchema        = false;
xmlFile = "../../../../../samples/data/auction.xml";
// Instantiate the DOM parser.
DOMParser parser;
parser.setValidationScheme(valScheme);
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
 parser.setErrorHandler(&errorHandler);
 unsigned long duration;
 try
`{
    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
    parser.parse(xmlFile);
    const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
duration = endMillis - startMillis;
}

catch (const XMLException& toCatch)
{
    cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
         << StrX(toCatch.getMessage()) << "\n" << endl;
    return -1;
}
catch (const DOM_DOMException& toCatch)
{
    cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
         << toCatch.code << "\n" << endl;
    XMLPlatformUtils::Terminate();
    return 4;
}
catch (...)
{
   cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
    XMLPlatformUtils::Terminate();
    return 4;
}
```

```
   if (errorHandler.getSawErrors())
      cout << "\nErrors occured, no output available\n" << endl;
    else
   {
      unsigned long staMil,endMil;
      cout<<xmlFile<<":"<<duration<<"ms"<<endl;
      staMil = XMLPlatformUtils::getCurrentMillis();
      DOM_Document doc = parser.getDocument();
      DOM_NodeList items=doc.getDocumentElement().getElementsByTagName("regions").
                                  getChildren(1).getChildren(1);
      unsigned int itemsCount = items.getLength();
      DOM_NodeList names=items.getChildren(1).filterTagName("name");
      endMil = XMLPlatformUtils::getCurrentMillis();
      duration = endMil - staMil;
      cout<<"processing time:"<<duration<<endl;
    }

   XMLPlatformUtils::Terminate();
   return 0;
   }
```

- The DOM solution

```
#define MAXLENGTH 1025
int main()
{
   try
      XMLPlatformUtils::Initialize();

   catch (const XMLException& toCatch)
   {
       cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
       return 1;
   }

   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                    doNamespaces    = false;
   bool                    doSchema        = false;
   xmlFile = "../../../../data/auction.xml";
   DOMParser parser;
   parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);
   DOMCountErrorHandler errorHandler;
   parser.setErrorHandler(&errorHandler);
   unsigned long duration;
   try
```

```
  {
     const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
     parser.parse(xmlFile);
     const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
     duration = endMillis - startMillis;
  }

  catch (const XMLException& toCatch)
  {
     cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
          << StrX(toCatch.getMessage()) << "\n" << endl;
         return -1;
  }
  catch (const DOM_DOMException& toCatch)
  {
     cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
          << toCatch.code << "\n" << endl;
     XMLPlatformUtils::Terminate();
     return 4;
  }
  catch (...)
  {
    cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
     XMLPlatformUtils::Terminate();
     return 4;
  }
if (errorHandler.getSawErrors())
     cout << "\nErrors occured, no output available\n" << endl;
else
{
  DOM_Element nameElem[MAXLENGTH];
  DOM_Document doc = parser.getDocument();
  DOM_NodeList regions=doc.getDocumentElement().getElementsByTagName("regions").
                                item(0).getChildNodes();
  unsigned int regCount = regions.getLength();
  unsigned int k=0;
  cout << xmlFile << ": " << duration << " ms ("<< regCount << " elems)." << endl;
  for(unsigned int i=0;i<regCount;i++)
  {
    DOM_Node regElem=regions.item(i);
     if(regElem.getNodeType()==1)
     {
       DOM_NodeList items=regElem.getChildNodes();
       unsigned int itemCount=items.getLength();
      for(unsigned int j=0;j<itemCount;j++)
       {
           DOM_Node itemElem=items.item(j);
           if(itemElem.getNodeType()==1)
```

```
            nameElem[k++]=(DOM_Element &)(((DOM_Element  &)itemElem).
                                getElementsByTagName("name").item(0));
        }
      }
    }
     cout<<"items' names length is:"<<k<<endl;
    }
    XMLPlatformUtils::Terminate();
    return 0;
    }
```

D:5 Source code of Example5

- The extension solution

```
//Get the number of bidders whose id is "person0"
int main()
{
 try
     XMLPlatformUtils::Initialize();

 catch (const XMLException& toCatch)
 {
    cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
    return 1;
 }

 const char*             xmlFile = 0;
    DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
    bool    doNamespaces    = false;
    bool    doSchema        = false;

    xmlFile = "../../../../../samples/data/auction.xml";
    // Instantiate the DOM parser.
    DOMParser parser;
    parser.setValidationScheme(valScheme);
    parser.setDoNamespaces(doNamespaces);
    parser.setDoSchema(doSchema);

    DOMCountErrorHandler errorHandler;
    parser.setErrorHandler(&errorHandler);
    unsigned long duration;
    try
    {
     const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
     parser.parse(xmlFile);
     const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
     duration = endMillis - startMillis;
    }
```

```
catch (const XMLException& toCatch)
{
  cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
       << StrX(toCatch.getMessage()) << "\n" << endl;
   return -1;
}
catch (const DOM_DOMException& toCatch)
{
   cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
       << toCatch.code << "\n" << endl;
   XMLPlatformUtils::Terminate();
   return 4;
}
catch (...)
{
  cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
   XMLPlatformUtils::Terminate();
   return 4;
}

if (errorHandler.getSawErrors())
   cout << "\nErrors occured, no output available\n" << endl;
 else
{
   unsigned long staMil,endMil;
   cout<<xmlFile<<":"<<duration<<"ms"<<endl;
   staMil = XMLPlatformUtils::getCurrentMillis();
   DOM_Document doc = parser.getDocument();
   DOM_NodeList open_auction=doc.getDocumentElement().
                                  getElementsByTagName("open_auctions").getChildren(1);
   unsigned int openCount = open_auction.getLength();
   cout << openCount << " elems)." << endl;
   DOM_NodeList bidders=open_auction.getChildren(1).filterTagName("bidder");
   cout<<"bidders length is "<<bidders.getLength()<<endl;
   DOM_NodeList personAttr=bidders.getChildren(1).
                                    filterTagName("personref").getAttributeList();
   DOM_NodeList person0=personAttr.filterValue("person0");
   unsigned int per0Count=person0.getLength();
   cout<<"person0 has "<<per0Count<<endl;
   endMil = XMLPlatformUtils::getCurrentMillis();
  duration = endMil - staMil;
  cout<<"processing time:"<<duration<<endl;
}
```

- The DOM solution

//Get the number of bidders whose id is "person0"

```
int main()
{
 try
    XMLPlatformUtils::Initialize();

 catch (const XMLException& toCatch)
 {
  cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
  return 1;
 }

const char*             xmlFile = 0;
DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
bool                    doNamespaces    = false;
bool                    doSchema        = false;
xmlFile = "../../../../data/auction.xml";
DOMParser parser;
parser.setValidationScheme(valScheme);
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);
unsigned long duration;
try
{
   const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
   parser.parse(xmlFile);
   const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
      duration = endMillis - startMillis;
   }

   catch (const XMLException& toCatch)
   {
      cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
           << StrX(toCatch.getMessage()) << "\n" << endl;
      return -1;
   }
   catch (const DOM_DOMException& toCatch)
   {
      cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
           << toCatch.code << "\n" << endl;
      XMLPlatformUtils::Terminate();
      return 4;
   }
   catch (...)
   {
     cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
      XMLPlatformUtils::Terminate();
```

```
      return 4;
   }

   if (errorHandler.getSawErrors())
      cout << "\nErrors occured, no output available\n" << endl;
  else
   {
    DOM_Document doc = parser.getDocument();
    DOM_NodeList opens=doc.getDocumentElement().getElementsByTagName("open_auctions").
                                item(0).getChildNodes();
    unsigned int openCount = opens.getLength();

    cout << xmlFile << ": " << duration << " ms ("<< openCount << " elems)." << endl;
    unsigned int bidsNum=0;
    for(unsigned int i=0;i<openCount;i++)
    {
     DOM_Node openElem=opens.item(i);
     if(openElem.getNodeType()==1)
     {
       DOM_NodeList bidders=((DOM_Element &)openElem).getElementsByTagName("bidder");
       unsigned int bidCount=bidders.getLength();
       for(unsigned int j=0;j<bidCount;j++)
      {
        DOM_Node bidElem=bidders.item(j);
        if(bidElem.getNodeType()==1)
        {
          DOMString person=((DOM_Element &)bidElem).getElementsByTagName("personref").
                                item(0).getAttributes().item(0).getNodeValue();
  if(person.equals("person0"))     bidsNum++;
   }
 }
}}
 cout<<"the number of the bidders whose id are person0 is "<<bidsNum<<endl;
}
XMLPlatformUtils::Terminate();
return 0;
}
```

D:6 Source code of Example6

- The extension solution

```
int main()
{
 try
      XMLPlatformUtils::Initialize();

 catch (const XMLException& toCatch)
 {
```

```
    cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
    return 1;
  }

const char*             xmlFile = 0;
DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
bool                     doNamespaces    = false;
bool                     doSchema        = false;
xmlFile = "../../../../../samples/data/auction.xml";
DOMParser parser;
parser.setValidationScheme(valScheme);
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);

DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);
unsigned long duration;
try
{
  const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
  parser.parse(xmlFile);
  const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
  duration = endMillis - startMillis;
}

catch (const XMLException& toCatch)
{
  cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
           << StrX(toCatch.getMessage()) << "\n" << endl;
       return -1;
    }
    catch (const DOM_DOMException& toCatch)
    {
        cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
            << toCatch.code << "\n" << endl;
        XMLPlatformUtils::Terminate();
        return 4;
    }
    catch (...)
    {
       cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
        XMLPlatformUtils::Terminate();
        return 4;
    }

    if (errorHandler.getSawErrors())
        cout << "\nErrors occured, no output available\n" << endl;
   else
```

```
  {
    unsigned long staMil,endMil;
    cout<<xmlFile<<":"<<duration<<"ms"<<endl;
    staMil = XMLPlatformUtils::getCurrentMillis();
    DOM_Document doc = parser.getDocument();
    DOM_NodeList open_auction=doc.getDocumentElement().
                        getElementsByTagName("open_auctions").getChildren(1);
    unsigned int openCount = open_auction.getLength();
    DOM_NodeList bidders=open_auction.getChildren(1).filterTagName("bidder");
    cout<<"bidders length is "<<bidders.getLength()<<endl;
    DOM_NodeList personAttr=bidders.getChildren(1).filterTagName("personref").
                                    getAttributeList();
    cout<<"personAttr="<<personAttr.getLength()<<endl;
    DOM_NodeList person0=personAttr.filterValue("person0");
    unsigned int per0Count=person0.getLength();
    cout<<"person0 has "<<per0Count<<endl;
    DOM_NodeList opens=person0.getParents().getParents().getParents().distinct(true);
    unsigned int opensCount=opens.getLength();
    cout<<"open_auction has "<<opensCount<<endl;

    endMil = XMLPlatformUtils::getCurrentMillis();
    duration = endMil - staMil;
    cout<<"processing time:"<<duration<<endl;
  }

  XMLPlatformUtils::Terminate();
  return 0;
}
```

- The DOM solution

```
#define MAXLENGTH 1025
//List those open auctions whose bidder's id is "person0"
int main()
{
 try
   XMLPlatformUtils::Initialize();

 catch (const XMLException& toCatch)
 {
  cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
  return 1;
 }

const char*             xmlFile = 0;
DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
bool                    doNamespaces    = false;
bool                    doSchema        = false;
```

```
xmlFile = "../../../../data/auction.xml";
DOMParser parser;
parser.setValidationScheme(valScheme);
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);
unsigned long duration;
try
{
 const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
 parser.parse(xmlFile);
 const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
 duration = endMillis - startMillis;
 }

 catch (const XMLException& toCatch)
 {
  cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
         << StrX(toCatch.getMessage()) << "\n" << endl;
  return -1;
 }
catch (const DOM_DOMException& toCatch)
{
 cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
         << toCatch.code << "\n" << endl;
 XMLPlatformUtils::Terminate();
 return 4;
}
catch (...)
{     cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
      XMLPlatformUtils::Terminate();
      return 4;
}

if (errorHandler.getSawErrors())
      cout << "\nErrors occured, no output available\n" << endl;
else
   {
      DOM_Document doc = parser.getDocument();
      DOM_NodeList opens=doc.getDocumentElement().
                        getElementsByTagName("open_auctions").item(0).getChildNodes();
      unsigned int openCount = opens.getLength();
      DOM_Element auction[MAXLENGTH];
      unsigned int k=0;
      cout << xmlFile << ": " << duration << " ms ("<< openCount << " elems)." << endl;
     for(unsigned int i=0;i<openCount;i++)
     {
```

```
        DOM_Node openElem=opens.item(i);
        if(openElem.getNodeType()==1)
        {
         DOM_NodeList bidders=((DOM_Element &)openElem).
                                    getElementsByTagName("bidder");
        unsigned int bidCount=bidders.getLength();
        for(unsigned int j=0;j<bidCount;j++)
        {
          DOM_Node bidElem=bidders.item(j);
          if(bidElem.getNodeType()==1)
         {
            DOMString person=((DOM_Element &)bidElem).getElementsByTagName("personref").
                                  item(0).getAttributes().item(0).getNodeValue();
            if(person.equals("person0"))
            {
              auction[k++]=(DOM_Element &)openElem;
              break;
            }
        }}}}
       cout<<"the selected open auctions have "<<k<<endl;
   }
   XMLPlatformUtils::Terminate();
   return 0;
}
```

D:7 Source code of Example7

- The extension solution

```
int main()
{
   try
      XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   {
       cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
       return 1;
   }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                    doNamespaces    = false;
   bool                    doSchema        = false;
   xmlFile = "../../../../../samples/data/auction.xml";
   DOMParser parser;
   parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);
   DOMCountErrorHandler errorHandler;
   parser.setErrorHandler(&errorHandler);
```

```
unsigned long duration;
try
{
    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
    parser.parse(xmlFile);
    const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
    duration = endMillis - startMillis;
}
catch (const XMLException& toCatch)
{
    cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
         << StrX(toCatch.getMessage()) << "\n" << endl;
    return -1;
}
catch (const DOM_DOMException& toCatch)
{
    cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
         << toCatch.code << "\n" << endl;
    XMLPlatformUtils::Terminate();
    return 4;
}
catch (...)
{
   cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
    XMLPlatformUtils::Terminate();
    return 4;
}

if (errorHandler.getSawErrors())
    cout << "\nErrors occured, no output available\n" << endl;
else
{
    unsigned long staMil,endMil;
    cout<<xmlFile<<":"<<duration<<"ms"<<endl;
    staMil = XMLPlatformUtils::getCurrentMillis();
    DOM_Document doc = parser.getDocument();
    DOM_NodeList siteChildren=doc.getDocumentElement().getChildNodes();
    DOM_NodeList personNodes=siteChildren.filterTagName("people").getChildren(1);
    unsigned int personCount = personNodes.getLength();
    DOM_NodeList idNodes=personNodes.getAttributeList();
     unsigned int idCount=idNodes.getLength();
    DOM_NodeList idTexts=idNodes.getChildren(0);
    unsigned int textCount=idTexts.getLength();
    cout << "personCount=" <<personCount<<",idCount="<<idCount
          << " textCount= "<<textCount<<"elems)." << endl;
   DOM_NodeList openNodes=siteChildren.filterTagName("open_auctions").getChildren(1);
   DOM_NodeList privacyNodes=openNodes.getChildren(1).filterTagName("privacy").
                                        getChildren(0).filterValue("Yes").getParents();
```

```
    DOM_NodeList opens=privacyNodes.getParents().distinct(true);
    DOM_NodeList bidderAttrs=opens.getChildren(1).filterTagName("bidder").
                        getChildren(1).filterTagName("personref").getAttributeList();
    DOM_NodeList bidderTexts=bidderAttrs.getChildren(0).distinct(true);
    unsigned int bidderCount=bidderTexts.getLength();
    cout<<"bidderAttrs length is "<<bidderAttrs.getLength()<<"bidderCount="
        <<bidderCount<<endl;
    DOM_NodeList ids=idTexts.intersect(bidderTexts,true);
    DOM_NodeList biddersName=ids.getParents().getParents().getChildren(1).
                                        filterTagName("name");
     endMil = XMLPlatformUtils::getCurrentMillis();
     duration = endMil - staMil;
     cout<<"processing time:"<<duration<<endl;
   }
   XMLPlatformUtils::Terminate();
   return 0;
   }
```

- The DOM solution

```
#define MAXLENGTH 1024
//List bidders' name nodes in the private open auctions.(join person and open auctions)
int main()
{
  try
    XMLPlatformUtils::Initialize();
    catch (const XMLException& toCatch)
    {cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
      return 1;
    }
    const char*             xmlFile = 0;
    DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
    bool                    doNamespaces    = false;
    bool                    doSchema        = false;
    xmlFile = "../../../../data/auction.xml";
    DOMParser parser;
    parser.setValidationScheme(valScheme);
    parser.setDoNamespaces(doNamespaces);
    parser.setDoSchema(doSchema);
    DOMCountErrorHandler errorHandler;
    parser.setErrorHandler(&errorHandler);
    unsigned long duration;
    try
    {
          const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
          parser.parse(xmlFile);
          const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
          duration = endMillis - startMillis;
    }
```

```
catch (const XMLException& toCatch)
{
   cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
        << StrX(toCatch.getMessage()) << "\n" << endl;
   return -1;
}
catch (const DOM_DOMException& toCatch)
{
  cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
           << toCatch.code << "\n" << endl;
  XMLPlatformUtils::Terminate();
  return 4;
}
catch (...)
{
   cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
   XMLPlatformUtils::Terminate();
   return 4;
}
if (errorHandler.getSawErrors())
  cout << "\nErrors occured, no output available\n" << endl;
else
{

       DOM_Document doc = parser.getDocument();
       DOM_Element siteElem=doc.getDocumentElement();
       DOM_NodeList pers=siteElem.getElementsByTagName("people").item(0).getChildNodes();
       unsigned int persCount = pers.getLength();
       DOM_NodeList opens=siteElem.getElementsByTagName("open_auctions").item(0).
                                    getChildNodes();
       unsigned int opensCount = opens.getLength();
       cout << xmlFile << ": " << duration << " ms ("<< opensCount << " elems)." << endl;
       DOM_Element name[MAXLENGTH];
       unsigned int m=0;
       bool mark[MAXLENGTH];
      for(unsigned int n=0;n<MAXLENGTH;n++)
                 mark[n]=false;

      for(unsigned int i=0;i<opensCount;i++)
     {  DOM_Node openElem=opens.item(i);
        if(openElem.getNodeType()==1)
        {DOM_NodeList privs=((DOM_Element &)openElem).getElementsByTagName("privacy");
          if(privs.getLength()>0)
          {
           DOM_Node privElem=privs.item(0);
           if(privElem.getFirstChild().getNodeValue().equals("Yes"))
           {
```

```
            DOM_NodeList bidders=((DOM_Element &)openElem).
                                    getElementsByTagName("bidder");
            unsigned int bidCount=bidders.getLength();
            for(unsigned int j=0;j<bidCount;j++)
            {
             DOM_Node bidElem=bidders.item(j);
             DOMString person_ref=((DOM_Element &)bidElem).
             getElementsByTagName("personref").item(0).getAttributes().item(0).getNodeValue();
            for(unsigned int k=0;k<persCount;k++)
            {
             DOM_Node personElem=pers.item(k);
             if(personElem.getNodeType()==1)
             {
             DOMString person_id=personElem.getAttributes().item(0).getNodeValue();
              if((person_ref.equals(person_id))&&(mark[k]==false))
              {name[m++]=(DOM_Element &)(((DOM_Element &)personElem).
                            getElementsByTagName("name").item(0));
               mark[k]=true;
              }
          }
         }
         }
     }
    }}}
   cout<<"the number of bidders is "<<m<<endl;
   }
   XMLPlatformUtils::Terminate();
   return 0;
}
```

D:8 Source code of Example8

- The extension solution

```
int main()
{
  try
        XMLPlatformUtils::Initialize();
  catch (const XMLException& toCatch)
  {  cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
     return 1;
  }
const char*             xmlFile = 0;
DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
bool                     doNamespaces    = false;
bool                     doSchema        = false;
xmlFile = "../../../../../samples/data/auction.xml";
DOMParser parser;
parser.setValidationScheme(valScheme);
```

```
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);
unsigned long duration;
try
{
   const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
   parser.parse(xmlFile);
   const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
   duration = endMillis - startMillis;
}
catch (const XMLException& toCatch)
{ cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
             << StrX(toCatch.getMessage()) << "\n" << endl;
   return -1;
}
catch (const DOM_DOMException& toCatch)
{ cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
             << toCatch.code << "\n" << endl;
   XMLPlatformUtils::Terminate();
   return 4;
}
catch (...)
    {
        cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
         XMLPlatformUtils::Terminate();
         return 4;
    }

    if (errorHandler.getSawErrors())
     cout << "\nErrors occured, no output available\n" << endl;
    else
      {
         unsigned long staMil,endMil;
         cout<<xmlFile<<":"<<duration<<"ms"<<endl;
         staMil = XMLPlatformUtils::getCurrentMillis();
         DOM_Document doc = parser.getDocument();
         DOM_Element siteElem=doc.getDocumentElement();
         DOM_NodeList personsid=siteElem.getElementsByTagName("people").
                              getChildren(1).getAttributeList().getChildren(0);
         unsigned int idCount = personsid.getLength();
         DOM_NodeList openNodes=siteElem.getElementsByTagName("open_auctions").
                                        getChildren(1);
         DOM_NodeList itemrefNodes=openNodes.getChildren(1).filterTagName("itemref").
                                           getAttributeList().getChildren(0);
         DOM_NodeList regions_africa=siteElem.getElementsByTagName("regions").
              getChildren(1).filterTagName("africa").getChildren(1).getAttributeList().getChildren(0);
```

```
        DOM_NodeList itemsinAfrica=itemrefNodes.intersect(regions_africa,true);
        DOM_NodeList biddersid=itemsinAfrica.getParents().getParents().getParents().
                getChildren(1).filterTagName("bidder").getChildren(1).filterTagName("personref").
                getAttributeList().getChildren(0);
        DOM_NodeList bidder_person=personsid.intersect(biddersid,true);
        DOM_NodeList biddersname=bidder_person.getParents().getParents().
                                        getChildren(1).filterTagName("name");
         unsigned int bidderCount=biddersname.getLength();
         cout<<"biddersname length is "<<bidderCount<<endl;
        endMil = XMLPlatformUtils::getCurrentMillis();
        duration = endMil - staMil;
        cout<<"processing time:"<<duration<<endl;
      }
      XMLPlatformUtils::Terminate();
      return 0;
   }
```

- The DOM solution

```
#define MAXLENGTH 1024
//Get the names of persons who bid on items from Africa.(Join person,open_auctions,item)
int main()
{
   try
   XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   {cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
    return 1;
    }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                     doNamespaces    = false;
   bool                     doSchema        = false;
   xmlFile = "../../../../data/auction.xml";
   DOMParser parser;
   parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);
   DOMCountErrorHandler errorHandler;
   parser.setErrorHandler(&errorHandler);
   unsigned long duration;
   try
   {
     const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
     parser.parse(xmlFile);
     const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
     duration = endMillis - startMillis;
   }
```

```
catch (const XMLException& toCatch)
 {cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
          << StrX(toCatch.getMessage()) << "\n" << endl;
      return -1;
  }
catch (const DOM_DOMException& toCatch)
{cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
          << toCatch.code << "\n" << endl;
  XMLPlatformUtils::Terminate();
  return 4;
}
catch (...)
{  cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
   XMLPlatformUtils::Terminate();
   return 4;
 }
if (errorHandler.getSawErrors())
      cout << "\nErrors occured, no output available\n" << endl;
else
{DOM_Document doc = parser.getDocument();
  DOM_Element siteElem=doc.getDocumentElement();
  DOM_NodeList persons=siteElem.getElementsByTagName("people").item(0).getChildNodes();
  unsigned int persCount = persons.getLength();
      DOM_NodeList opens=siteElem.getElementsByTagName("open_auctions").
                                  item(0).getChildNodes();
      unsigned int opensCount = opens.getLength();
      DOM_NodeList items=((DOM_Element &)(siteElem.getElementsByTagName("regions").
                          item(0))).getElementsByTagName("africa").item(0).getChildNodes();
       unsigned int itemCount = items.getLength();
       cout << xmlFile << ": " << duration << " ms " << endl;
       DOM_Element person_name[MAXLENGTH];
       unsigned int m=0;
       bool mark1[MAXLENGTH],mark2[MAXLENGTH];
       for(unsigned int n=0;n<MAXLENGTH;n++)
       {
       mark1[n]=false;
       mark2[n]=false;
       }
       for(unsigned int i=0;i<opensCount;i++)
       {
       DOM_Node openElem=opens.item(i);
       if(openElem.getNodeType()==1)
       {DOMString item_id=((DOM_Element &)openElem).getElementsByTagName("itemref").
                                  item(0).getAttributes().item(0).getNodeValue();
         for(unsigned int j=0;j<itemCount;j++)
         {DOM_Node itemElem=items.item(j);
           if(itemElem.getNodeType()==1)
          {DOMString item_africa=itemElem.getAttributes().item(0).getNodeValue();
```

```
            if((item_id.equals(item_africa))&&(mark1[j]==false))
            {  mark1[j]=true;
               DOM_NodeList bidders=((DOM_Element &)openElem).
                                        getElementsByTagName("bidder");
              for(unsigned int l=0;l<bidders.getLength();l++)
              {DOMString bidder_id=((DOM_Element &)(bidders.item(l))).
               getElementsByTagName("personref").item(0).getAttributes().item(0).getNodeValue();
              for(unsigned int k=0;k<persCount;k++)
              {DOM_Node personElem=persons.item(k);
                if(personElem.getNodeType()==1)
                {if((personElem.getAttributes().item(0).getNodeValue().equals(bidder_id))
                    &&(mark2[k]==false))
                   {mark2[k]=true;
                    person_name[m++]=(DOM_Element &)(((DOM_Element &)personElem).
                                        getElementsByTagName("name").item(0));
                    break; }
                 }
               }
          }
         }
         }
         }}
         }

      cout<<"the number of persons is"<<m<<endl;
    }
   XMLPlatformUtils::Terminate();
   return 0;
}
```

D:9 Source code of Example9

- The extension solution

```
int main()
{
   try
      XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   {cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
    return 1;
   }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                    doNamespaces    = false;
   bool                    doSchema        = false;
   xmlFile = "../../../../../samples/data/auction.xml";
   DOMParser parser;
   parser.setValidationScheme(valScheme);
```

```
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);
unsigned long duration;
try
{
      const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
      parser.parse(xmlFile);
      const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
      duration = endMillis - startMillis;
 }
catch (const XMLException& toCatch)
{cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
         << StrX(toCatch.getMessage()) << "\n" << endl;
 return -1;}
 catch (const DOM_DOMException& toCatch)
 {cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
         << toCatch.code << "\n" << endl;
      XMLPlatformUtils::Terminate();
      return 4;}
 catch (...)
 {cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
   XMLPlatformUtils::Terminate();
      return 4;
    }
 if (errorHandler.getSawErrors())
  cout << "\nErrors occured, no output available\n" << endl;
 else
 {
   unsigned long staMil,endMil;
   cout<<xmlFile<<":"<<duration<<"ms"<<endl;
   staMil = XMLPlatformUtils::getCurrentMillis();
   DOM_Document doc = parser.getDocument();
   DOM_Element siteElem=doc.getDocumentElement();
   DOM_NodeList closed_auction=siteElem.getElementsByTagName("closed_auctions").
                    getChildren(1).getChildren(1);
   unsigned int closeCount = closed_auction.getLength();
   DOM_NodeList sellers=closed_auction.filterTagName("seller").
                                getAttributeList().getChildren(0);
   DOM_NodeList buyers=closed_auction.filterTagName("buyer").
                                getAttributeList().getChildren(0);
   DOM_NodeList dis_seller=sellers.subtract(buyers,true);
    endMil = XMLPlatformUtils::getCurrentMillis();
    duration = endMil - staMil;
    cout<<"processing time:"<<duration<<endl;
    }
  XMLPlatformUtils::Terminate();
```

```
    return 0;
  }
```

- The DOM solution

```
#define MAXLENGTH 1024
//List all of persons id who are only sellers and do not buy other items in closed auctions.
int main()
{
 try
    XMLPlatformUtils::Initialize();
  catch (const XMLException& toCatch)
   { cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
     return 1;
    }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                     doNamespaces    = false;
   bool                     doSchema        = false;
   xmlFile = "../../../../data/auction.xml";
   DOMParser parser;
   parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);
   DOMCountErrorHandler errorHandler;
   parser.setErrorHandler(&errorHandler);
   unsigned long duration;
   try
    {
         const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
         parser.parse(xmlFile);
         const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
         duration = endMillis - startMillis;
      }
    catch (const XMLException& toCatch)
    {cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
              << StrX(toCatch.getMessage()) << "\n" << endl;
         return -1;}
     catch (const DOM_DOMException& toCatch)
     { cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
              << toCatch.code << "\n" << endl;
         XMLPlatformUtils::Terminate();
         return 4;
      }
      catch (...)
      {
        cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
         XMLPlatformUtils::Terminate();
```

```
    return 4;
  }
  if (errorHandler.getSawErrors())
    cout << "\nErrors occured, no output available\n" << endl;
  else
  {DOM_Document doc = parser.getDocument();
   DOM_Element siteElem=doc.getDocumentElement();
   DOM_NodeList closed_auctions=siteElem.getElementsByTagName("closed_auctions").
                                          item(0).getChildNodes();
   unsigned int closedCount = closed_auctions.getLength();
   DOM_Node seller_id[MAXLENGTH],id[MAXLENGTH];
   unsigned int k=0,n=0,i,j;
   bool mark[MAXLENGTH];
   DOM_Node sellerAttr;
    cout << xmlFile << ": " << duration << " ms ("<<endl;
    for( i=0;i<closedCount;i++)
    {  DOM_Node closedElem=closed_auctions.item(i);
       if(closedElem.getNodeType()==1)
       {DOM_Node seller=((DOM_Element &)closedElem).
                                  getElementsByTagName("seller").item(0);
        for(j=0;j<closedCount;j++)
        {DOM_Node closedNode=closed_auctions.item(j);
         if(closedNode.getNodeType()==1)
         {DOM_Node buyer=((DOM_Element &)(closedNode)).
                                   getElementsByTagName("buyer").item(0);
          sellerAttr=seller.getAttributes().item(0);
          DOMString seller_id=sellerAttr.getNodeValue();
          DOMString buyer_id=buyer.getAttributes().item(0).getNodeValue();
          if(seller_id.equals(buyer_id))   break;
         }
        }
       if(j==closedCount) seller_id[k++]=sellerAttr;
       }}
       for(i=0;i<k;i++)
           mark[i]=false;
       for(i=0;i<k;i++)
       {if(mark[i]==false)
        { id[n++]=seller_id[i];
          for(j=i+1;j<k;j++)
          {
           if(seller_id[i].getNodeValue().equals(seller_id[j].getNodeValue()))
           mark[j]=true;
          }
        }
       }
    cout<<"the number of sellers is "<<n<<endl;
  }
XMLPlatformUtils::Terminate();
```

```
    return 0;
}
```

D:10 Source code of Example10

- The extension solution

```
//List those open auctions on which all buyers watch according to document order
int main()
{
   try
        XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
    {  cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
       return 1;
     }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                    doNamespaces    = false;
   bool                    doSchema        = false;
   xmlFile = "../../../../../samples/data/auction.xml";
   DOMParser parser;
  parser.setValidationScheme(valScheme);
   parser.setDoNamespaces(doNamespaces);
   parser.setDoSchema(doSchema);
   DOMCountErrorHandler errorHandler;
   parser.setErrorHandler(&errorHandler);
   unsigned long duration;
   try
    {
        const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
        parser.parse(xmlFile);
        const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
        duration = endMillis - startMillis;
     }
   catch (const XMLException& toCatch)
    {cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
             << StrX(toCatch.getMessage()) << "\n" << endl;
        return -1;
      }
    catch (const DOM_DOMException& toCatch)
     {cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
             << toCatch.code << "\n" << endl;
        XMLPlatformUtils::Terminate();
        return 4;
      }
    catch (...)
    {   cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
        XMLPlatformUtils::Terminate();
```

```
        return 4;
    }
   if (errorHandler.getSawErrors())
        cout << "\nErrors occured, no output available\n" << endl;
   else
    {
     unsigned long staMil,endMil;
     cout<<xmlFile<<":"<<duration<<"ms"<<endl;
     staMil = XMLPlatformUtils::getCurrentMillis();
     DOM_Document doc = parser.getDocument();
     DOM_Element siteElem=doc.getDocumentElement();
     DOM_NodeList personsid=siteElem.getElementsByTagName("people").
                          getChildren(1).getAttributeList().getChildren(0);
     DOM_NodeList buyers=siteElem.getElementsByTagName("closed_auctions").
                                  getChildren(1).getChildren(1).filterTagName("buyer").
                                  getAttributeList().getChildren(0).distinct(true);
     DOM_NodeList buyer_persons=personsid.intersect(buyers,true);
     DOM_NodeList opens_id=buyer_persons.getParents().getParents().getChildren(1).
              filterTagName("watches").getChildren(1).getAttributeList().getChildren(0);
     DOM_NodeList open_auction=siteElem.getElementsByTagName("open_auctions").
                         getChildren(1).getAttributeList().getChildren(0);
     DOM_NodeList result_auctions=open_auction.intersect(opens_id,true).
                                          getParents().getParents().sort(true);
     cout << "result_auctions="<<result_auctions.getLength()<<"elems)." << endl;
          endMil = XMLPlatformUtils::getCurrentMillis();
          duration = endMil - staMil;
          cout<<"processing time:"<<duration<<endl;
       }
     XMLPlatformUtils::Terminate();
      return 0;
   }
```

- the DOM solution

```
#define MAXLENGTH 1024
//List those open auctions on which all buyers watch according to document order
int main()
{
  try
      XMLPlatformUtils::Initialize();
   catch (const XMLException& toCatch)
   {
       cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
       return 1;
   }
   const char*             xmlFile = 0;
   DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
   bool                    doNamespaces    = false;
```

```
bool  doSchema        = false;
xmlFile = "../../../../data/auction.xml";
DOMParser parser;
parser.setValidationScheme(valScheme);
parser.setDoNamespaces(doNamespaces);
parser.setDoSchema(doSchema);
DOMCountErrorHandler errorHandler;
parser.setErrorHandler(&errorHandler);
unsigned long duration;
try
{
       const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
       parser.parse(xmlFile);
       const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
       duration = endMillis - startMillis;
}
catch (const XMLException& toCatch)
{cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
           << StrX(toCatch.getMessage()) << "\n" << endl;
       return -1;
   }
catch (const DOM_DOMException& toCatch)
{cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
           << toCatch.code << "\n" << endl;
       XMLPlatformUtils::Terminate();
       return 4;
}
catch (...)
{cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
  XMLPlatformUtils::Terminate();
  return 4;}
if (errorHandler.getSawErrors())
    cout << "\nErrors occured, no output available\n" << endl;
else
{DOM_Document doc = parser.getDocument();
  DOM_Element siteElem=doc.getDocumentElement();
  DOM_NodeList closed_auctions=siteElem.getElementsByTagName("closed_auctions").
                                        item(0).getChildNodes();
  unsigned int closedCount = closed_auctions.getLength();
  DOM_NodeList open_auctions=siteElem.getElementsByTagName("open_auctions").
                                      item(0).getChildNodes();
  unsigned int openCount = open_auctions.getLength();
  DOM_NodeList persons=siteElem.getElementsByTagName("people").item(0).getChildNodes();
  unsigned int personCount = persons.getLength();
  DOM_Node openAuct[MAXLENGTH];
  unsigned int n=0;
  unsigned int i,j,k,m;
  bool mark[MAXLENGTH];
```

```
cout << xmlFile << ": " << duration << " ms ("<<endl;
for(i=0;i<openCount;i++)
      mark[i]=false;
for( i=0;i<openCount;i++)
{DOM_Node openElem=open_auctions.item(i);
 mark[i]=false;
 if(openElem.getNodeType()==1)
 {DOMString open_id=openElem.getAttributes().item(0).getNodeValue();
  for(j=0;j<closedCount;j++)
  { DOM_Node closedElem=closed_auctions.item(j);
    if(closedElem.getNodeType()==1)
    {DOM_Node buyer=((DOM_Element &)(closedElem)).getElementsByTagName("buyer").
                                item(0);
     DOMString buyer_id=buyer.getAttributes().item(0).getNodeValue();
     for(k=0;k<personCount;k++)
      {DOM_Node personElem=persons.item(k);
       if(personElem.getNodeType()==1)
       {DOMString person_id=personElem.getAttributes().item(0).getNodeValue();
        if(buyer_id.equals(person_id))
        {DOM_NodeList watchesList=((DOM_Element &)personElem).
                                          getElementsByTagName("watches");
         if(watchesList.getLength()>0)
         {DOM_NodeList watches=watchesList.item(0).getChildNodes();
          unsigned int watchesCount=watches.getLength();
              for(m=0;m<watchesCount;m++)
              {DOM_Node watchElem=watches.item(m);
               if(watchElem.getNodeType()==1)
               {DOMString watch_id=watchElem.getAttributes().item(0).getNodeValue();
                if((open_id.equals(watch_id))&&(mark[i]==false))
                {openAuct[n++]=openElem;
                 mark[i]=true;
                 break;}
               }}
              }}}}}}}}
              cout<<"the number of open auctions is "<<n<<endl;
     }
  XMLPlatformUtils::Terminate();
  return 0;}
```

D:11 Source code of Example11

- The extension solution

```
//Get the first 5 and the last 5 closed auctions in document order, for which the seller does ont buy
//other items in closed auctions, return the seller, buyer, item and price.
int main()
{
 try
    XMLPlatformUtils::Initialize();
```

```
catch (const XMLException& toCatch)
{cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
 return 1;
 }
 const char*             xmlFile = 0;
 DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
 bool                     doNamespaces    = false;
 bool                     doSchema        = false;
 xmlFile = "../../../../../samples/data/auction2.xml";
 DOMParser parser;
  parser.setValidationScheme(valScheme);
  parser.setDoNamespaces(doNamespaces);
  parser.setDoSchema(doSchema);
  DOMCountErrorHandler errorHandler;
  parser.setErrorHandler(&errorHandler);
  unsigned long duration;
  try
  { const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
     parser.parse(xmlFile);
     const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
     duration = endMillis - startMillis;
  }
  catch (const XMLException& toCatch)
  {cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
         << StrX(toCatch.getMessage()) << "\n" << endl;
     return -1;
  }
  catch (const DOM_DOMException& toCatch)
  {cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
         << toCatch.code << "\n" << endl;
     XMLPlatformUtils::Terminate();
     return 4;}
  catch (...)
  { cerr << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
     XMLPlatformUtils::Terminate();
     return 4;  }
  if (errorHandler.getSawErrors())
    cout << "\nErrors occured, no output available\n" << endl;
  else
  {unsigned long staMil,endMil;
   cout<<xmlFile<<":"<<duration<<"ms"<<endl;
   staMil = XMLPlatformUtils::getCurrentMillis();
   DOM_Document doc = parser.getDocument();
   DOM_NodeList siteChilds=doc.getDocumentElement().createNodeList(1).getChildren(1);
   DOM_NodeList closed_auction=siteChilds.filterTagName("closed_auctions").
                                          getChildren(1).getChildren(1);
   DOM_NodeList sellers=closed_auction.filterTagName("seller").
                               getAttributeList().getChildren(0);
```

```
    DOM_NodeList buyers=closed_auction.filterTagName("buyer").
                                  getAttributeList().getChildren(0);
    DOM_NodeList final_sellers=sellers.subtract(buyers,true).getParents().getParents();
    DOM_NodeList closed_auctions=final_sellers.getParents();
    DOM_NodeList final1=closed_auctions.sort(false);
    unsigned int finlen=final1.getLength();
    DOM_NodeList final_auctions;
    if(finlen<=5) final_auctions=final1.subList(0,finlen);
    else
    {DOM_NodeList final1_auctions=final1.subList(0,5);
     if(finlen<10)
    DOM_NodeList final2_auctions=final1.subList(5,finlen-5).sort(true);
    final_auctions=final1_auctions.catenate(final2_auctions);}
    DOM_NodeList finals=final_auctions.getChildren(1);
    final_sellers=finals.filterTagName("seller");
    DOM_NodeList final_prices=finals.filterTagName("price");
    DOM_NodeList final_buyers=finals.filterTagName("buyer");
    DOM_NodeList final_items=finals.filterTagName("itemref");
    cout<<"final_sellers:"<<final_sellers.getLength()<<endl;
    cout<<"final_prices:"<<final_prices.getLength()<<endl;
    cout<<"final_buyers:"<<final_buyers.getLength()<<endl;
    cout<<"final_items:"<<final_items.getLength()<<endl;
    endMil = XMLPlatformUtils::getCurrentMillis();
    duration = endMil - staMil;
      cout<<"processing time:"<<duration<<endl;
    }
   XMLPlatformUtils::Terminate();
   return 0;
}
```

- The DOM solution

```
#define MAXLENGTH 1024
//Get the first 5 and last 5 closed auctions in document order, for which the seller does ont buy other
//items in closed auctions,return the seller, buyer, item and price.
int main()
{
 try
   XMLPlatformUtils::Initialize();
 catch (const XMLException& toCatch)
 {cerr << "Error during initialization! :\n"<< StrX(toCatch.getMessage()) << endl;
  return 1;
 }
 const char*             xmlFile = 0;
 DOMParser::ValSchemes    valScheme = DOMParser::Val_Never;
 bool                    doNamespaces    = false;
 bool                    doSchema        = false;
 xmlFile = "../../../../data/auction.xml";
```

```
  DOMParser parser;
  parser.setValidationScheme(valScheme);
  parser.setDoNamespaces(doNamespaces);
  parser.setDoSchema(doSchema);
  DOMCountErrorHandler errorHandler;
  parser.setErrorHandler(&errorHandler);

  unsigned long duration;
  try
  {    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
       parser.parse(xmlFile);
       const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
       duration = endMillis - startMillis;
  }
 catch (const XMLException& toCatch)
 {cerr << "\nError during parsing: '" << xmlFile << "'\n"<< "Exception message is:  \n"
          << StrX(toCatch.getMessage()) << "\n" << endl;
   return -1;
 }
catch (const DOM_DOMException& toCatch)
{cerr << "\nDOM Error during parsing: '" << xmlFile << "'\n"<< "DOMException code is:  \n"
          << toCatch.code << "\n" << endl;
  XMLPlatformUtils::Terminate();
  return 4;}
catch (...)
{err << "\nUnexpected exception during parsing: '" << xmlFile << "'\n";
  XMLPlatformUtils::Terminate();
  return 4;}

if (errorHandler.getSawErrors())
    cout << "\nErrors occured, no output available\n" << endl;
else
{DOM_Document doc = parser.getDocument();
  DOM_Element siteElem=doc.getDocumentElement();
  DOM_NodeList closed_auctions=siteElem.getElementsByTagName("closed_auctions").
                                        item(0).getChildNodes();
  unsigned int closedCount = closed_auctions.getLength();
  DOM_Node closedElem,sellerElem,closedNode,buyerElem;
  DOM_Node seller[MAXLENGTH],buyer[MAXLENGTH],
                item[MAXLENGTH],price[MAXLENGTH];
  DOM_Node selRes[20],buyRes[20],itemRes[20],pricRes[20];
  unsigned int k=0;
  unsigned int n=0;
  unsigned int i,j;

   cout << xmlFile << ": " << duration << " ms ("<< endl;
   for( i=0;i<closedCount;i++)
  {closedElem=closed_auctions.item(i);
```

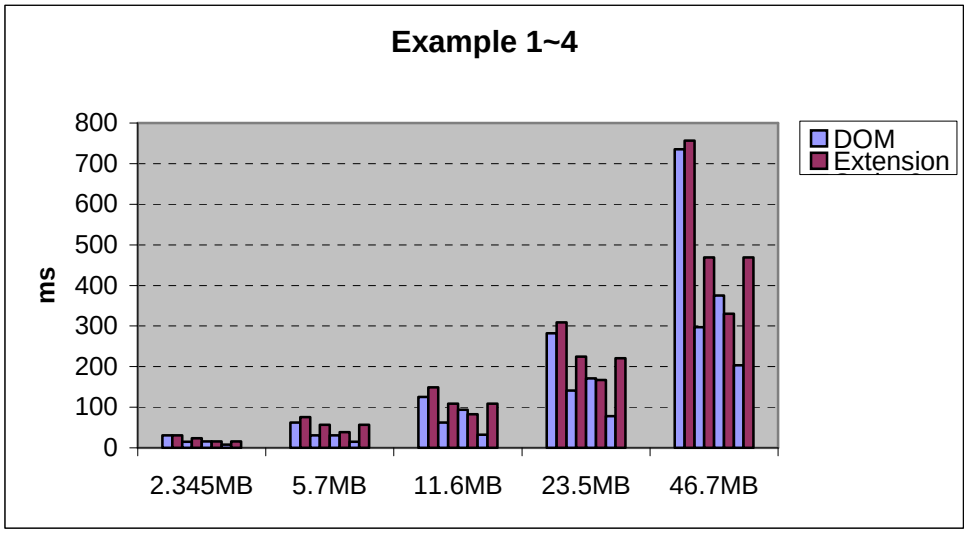
```
   if(closedElem.getNodeType()==1)
  {sellerElem=((DOM_Element &)closedElem).getElementsByTagName("seller").item(0);
   for(j=0;j<closedCount;j++)
   {closedNode=closed_auctions.item(j);
    if(closedNode.getNodeType()==1)
   {buyerElem=((DOM_Element &)(closedNode)).getElementsByTagName("buyer").item(0);
    DOMString seller_id=sellerElem.getAttributes().item(0).getNodeValue();
    DOMString buyer_id=buyerElem.getAttributes().item(0).getNodeValue();
     if(seller_id.equals(buyer_id))   break; }
    }
    if(j==closedCount)
    {seller[k]=sellerElem;
     buyer[k]=((DOM_Element &)(closedElem)).getElementsByTagName("buyer").item(0);
     item[k]=((DOM_Element &)(closedElem)).getElementsByTagName("itemref").item(0);
     price[k]=((DOM_Element &)(closedElem)).getElementsByTagName("price").item(0);
     k++; }
     }}
    if(k<5) n=k; else n=5;
    for(i=0;i<n;i++)
    {selRes[i]=seller[i];
     buyRes[i]=buyer[i];
     itemRes[i]=item[i];
     pricRes[i]=price[i]; }
     if(k>=5)
          { if((k>5)&&(k<10)) n=k-5; else n=10;
            for(i=0;i<n;i++)
           {selRes[5+i]=seller[k-1-i];
             buyRes[5+i]=buyer[k-1-i];
             itemRes[5+i]=item[k-1-i];
             pricRes[5+i]=price[k-1-i];
           }}
   cout<<"the number of the required close auctions is "<<k<<endl;
   }
   XMLPlatformUtils::Terminate();
   return 0;
}
```

Appendix F

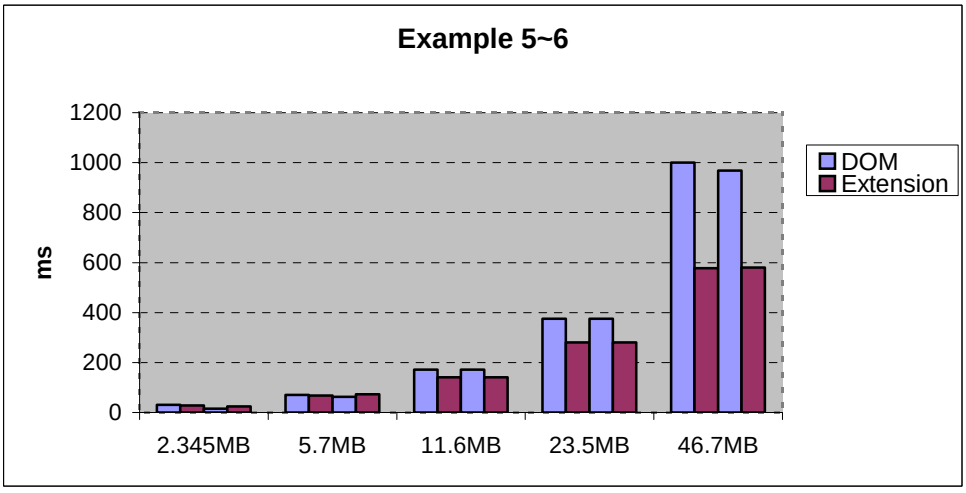
Comparison graphs

This appendix contains the comparison graph of examples used in this thesis based on the DOM and the extension with different size of data.

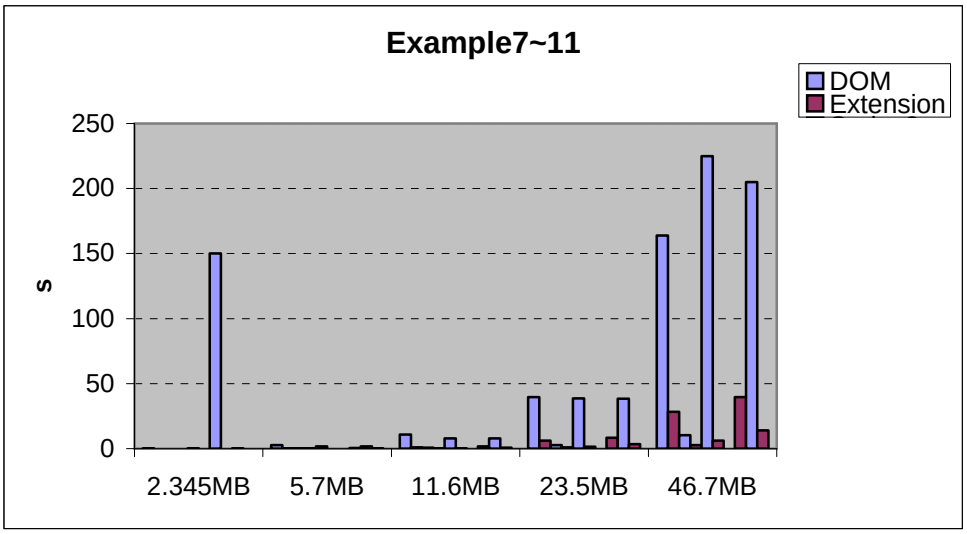
E.1: The comparison graph of example 1~4



E.2: The comparison graph of example 5~6



E.3: The comparison graph of example 7~11



Bibliography

[ABC00] Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, Lauren Wood. *Document Object Model Level 1 (Second Edition)*, Available at http://www.w3.org/TR/DOMTR, 2000.

[ABS00] S. Abiteboul, Peter Buneman, Dan Suciu, *Data on the Web*, Morgan Kaufmann, 2000.

[BBC03] Anders Berglund , Scott Boag , Don Chamberlin, *XML Path Language (XPath) 2.0*, Available at http://www.w3.org/TR/WD-xpath20-20030502/, 2003.

[BiM01] Paul V. Biron, Ashok Malhotra, *XML Schema Part 2: Datatypes*, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/, 2001.

[BoC03] Scott Boag, Don Chamberlin, *An XML Query Language*, Avaialble at http://www.w3.org/TR/2003/WD-xquery-20030502/ , 2003.

[BPS+00] T.Bray, J.Paoli, C.M. Sperberg-McQueen, and E.Maler. *Extensible Markup Language(XML)* 1.0(Second Edition). Available at http://www.w3.org/TR/REC-xml, 2000.

[BoR01] Timo Böhme, Erhard Rahm. *XMach-1: A Benchmark for XML Data Management*, Proceedings of BTW2001, Oldenburg, 7.-9. März, Springer, Berlin 2001, Available at http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html.

[CFM+01] Don Chamberlin, Peter Fankhauser, Massimo Marchiori, Johathan Robie, *XML Query Use Cases*, Available at http://www.w3.org/TR/2001/WD-xmlquery-use-cases-20010215, 2001.

[CGG+85] Bruce W.Char, Keith O.Geddes, Gaston H. Gonnet, Stephen M.Watt, *Maple user's guide* 1985.

[DeS86] N.M. Delisle, M.D. Schwartz, *A hypertext system for CAD applications*. In Proceedings of the International Conference on Management of Data (Washington, D.C.,May28-30,1986). ACM, New York, 1986, pp.132-143.

[FaI68] A.D. Falkof, K.E lverson. *APL\360 Users Manual*, IBM T.J. Watson Research Center, Yorktown, 1968.

[Fal01] David C. Fallside, *XML Schema Part 0: Primer*, Available at http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/, 2001.

[Gel89] Joseph R. Geller, *IMS administration, programming, and data base design*, New York ; Toronto : J. Wiley, 1989.

[Gra93] J. Gray. *Database and Transaction Processing Performance Handbook.* http://www.benchmarkresources.com/handbook, 1993.

[Heg03] Phillippe Le Hégaret, *Document Object Model (DOM) activity statement*, http://www.w3.org/DOM/Activity,2003.

[HHW+03] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, Steve Byrne. *Document Object Model (DOM) Level 3 Core Specification*, http://www.w3.org/TR/2003/WD-DOM-Level-3-Core-20030226, 2003.

[IEC00] The International Engineering Consortium, *Specification and Description Language(SDL)*, Web Proforum Tutorials, http://www.iec.org, 2000.

[Ive62] K.E lverson, *A programming language*, John Wiley & Sons, 1962

[Kor86] Henry F.Korth, *Database system concepts*, The McGraw-Hill Companies, Inc,1986.

[LLO+91] Charles Lamb, Gordon Landis, Jack Orenstein, Dan Weinreb, *The Objectstore Database System*, Communications of the ACM, 1991.

[Meg02] David Megginson, http://www.saxproject.org, 2002.

[MichiganMark websites] The Michigan Benchmark http://www.eecs.umich.edu/db/mbench.

[MMR+02] Ashok Malhotra, Jim Melton, Jonathan Robie, Norman Walsh, *XQuery 1.0 and XPath 2.0 Functions and Operators*, Available at http://www.w3.org/TR/2002/WD-xquery-operators-20020430/, 2002.

[OMGa02] Object Management Group, *CORBA™/IIOP™ Specification*, http://www.omg.org/cgi-bin/doc?formal/02-12-02 , 2002.

[OMGb02] Object Management Group, *The Common Object Request Broker: Architecture and Specification 2-2*, http://www.omg.org/ , 2002.

[OMGc02]Object Management Group, *The Common Object Request Broker: Architecture and Specification 2-3*, http://www.omg.org/ , 2002

[OMGa03]Object Management Group, *C++ Language Mapping Specification*, http://www.omg.org/docs/formal/03-06-03.pdf, 2003.

[OMGb03] Object Management Group, *OMG Unified Modeling Language Specification*, http://www.omg.org/uml, 2003.

[Pee86] Howard A. Peelle, *APL an introduction*, Sams, 1986.

[RaB02] E. Böhme T.Rahm, *XMach-1: A Multi-User Benchmark for XML Data Management.* Proc. VLDB workshop Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT2002), Hongkong, Aug. 2002.

[SHH03]Johnny Stenback, Philippe Le Hégaret, Arnaud Le Hors, *Document Object Model (DOM) Level 2 HTML Specification*, http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109, 2003.

[SWK+01] A.R.Schmidt, F.Waas, M.L. Kersten, D. Florescu, *The XML benchmark project*, Information Systems(INS), INS-R0103 April 30, Available at http://monetdb.cwi.nl/xml/index.html, 2001

[TBM+01] Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, *XML Schema Part 1: Structures*, Available at http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/, 2001.

[Tom89] Frank WM. Tompa, *A data model for flexible hypertext database systems*, ACM Transactions on Information Systems, Vol. 7, No. 1, January 1989, Pages 85-100, 1989.

[Ull80] Jefferey D.Ullman, *Principle of Database System*, W H Freeman & Co., 1980.

[X007 website] The XOO7 Benchmark http://www.comp.nus.edu.sg/~ebh/XOO7.html.

[Xbench website] http://db.uwaterloo.ca/~ddbms/projects/xbench/.

[Xerces website] http://xml.apache.org/xerces-c/index.html.

[XMark website] http://monetdb.cwi.nl/xml/index.html.

[XMach website] http://dbs.uni-leipzig.de.