Convert Object-Oriented Program (OOP) Into Extensible Markup Language (XML)

Prof. Dr. Hilal M. Yousif
Hadeel Sh. Kareem

Abstract:

Extensible Markup Language (XML) is a coding language that describes the structure of document and meaning, and it used to represent Object -Oriented-Programming (OOP) as Database document files such as Document Type Declaration (DTD) in textual way.

In this paper Reverse Engineering (RE) is used to analysis and document input Object-Oriented-Program (OOP) with (*.java; *.class) extension .After this point the Java XML (JXML) parser used to analysis document data (that it is a result from RE step) to XML tree (TAX) Structure that represent Document data and finally the Extensible Stylesheet Language Transformation (XSLT) used to generate XML files and DTD associative files.

XML is a joint effort to create a genuinely open standard, driven entirely by user needs. These needs include:

- ***Extensibility:* -** to define new tags as needed.
- ***Structure:* -**to model data to any level of complexity.
- ***Validation:* -**to check data for structural correctness.
- ***Media independence:* -** to publish content in multiple formats.
- ***Vendor and platform:* -** independence, to process any conforming document using standard commercial software or even simple text tools.

Keywords:

Object-Oriented-Design (OOD), Reverse Engineering (RE), Computer Aided Software Engineering (CASE), Extensible Markup Language (XML).

تحويل البرمجة الشيئية إلى
Extensible Markup Language (XML)

أ.د.هلال محمد يوسف
هديل شوكت كريم

المستخلص:

أن صيغة ألـ XML تعمل هيكلية تصميم على شكل قاعدة بيانات لذا نحن نستطيع تمثيل ألـ OOPعلى شكل توثيق لقاعدة البيانات وتعتمد علتعريف توثيق النوع(DTD)وبشكل نصي.

في هذا البحث تم استخدام الهندسة العكسية لتحليل وتوثيق البرمجة الشيئية (OOP) مع امتداد (*.java ;*.class) وثم العمل على أل (Java XML Parser) لتحويل نتائج و توثيقات الهندسة العكسية إلى تمثيل شجري يعتمد على صيغة ألـ(XML)وهو أل (TAX)والذي يستخدم في تمثيل عناصر النظام. وثم العمل على الـ(XSLT) لغرض تحويل إلى ملفات أل XML(*.xmi)و توثيقات أل Document Type Declaration(DTD)

أن أل XMLتقوم بخلق معيار أساسي لاحتياجات المستخدم وهذه الاحتياجات هي:-

- قابلية التوسع: لتعريف tagsجديد و إضافية.
- الهيكلية:- لنمذجة البيانات لأي مستوى من التعقيد.
- المشروعية :- لفحص البيانات لهيكلية صحيحة .
- بيئة مستقلة:- لنشر المحتوى بصيغ متعددة.
- لمعالجة أي توثيق باستخدام برمجيات معيارية واقتصادية أو أدوات نصية بسيطة.

1. Introduction

Object-Oriented-Programming is concerned with realizing a software design using an Object-Oriented-Programming language. An Object-Oriented-Programming language supports the direct implementation of objects and provides facilities to define object classes.

OO languages like Smalltalk, C++, and Java provide mechanisms to make the implementation easier [2,6].

The Object-Orientation systems using the Reverse Engineering (RE) to involve analyzing a system to identify its components and interrelationships and create representations of the system in another form or at a higher level of abstraction.

Extensible Markup Language (XML) is a coding language that describes the structure of document and meaning, and it used to represent Object-Oriented-Programming (OOP) as Database document files such as Document Type Declaration (DTD) in textual way.

In this paper we suggest systems that convert Object-Oriented Program system (OOP) into XML schema with DTD files. This system acceptance OOP system with (*.java ;*.class) extension and it used reverse engineering as a tool to analysis and document an input system and the proposed system was used Java XML (JXML) parser to read document data which is a result from RE step and analysis it to generate Algebra Tree XML (TAX) and finally we used the Extensible Stylesheet Language Transformation(XSLT) to translate TAX into XML files(*.xmi) and Document Type Declaration (DTD) associative files.

In this paper XML is designed to store, transmit and exchange data, XML can encode the representation for:

- An ordinary document
- A structured (database) record, such as an appointment record
- An object, with data and methods (e.g. a Java object or an ActiveX control).
- Graphical presentation (such as an application's user interface).

2. Object-Oriented-Programming

Object-Oriented-Programming (OOP) is concerned with realizing a software design using an Object-Oriented-Programming language. An OOP language supports the direct implementation of objects and provides facilities to define object classes. OOP is, at its highest level, the process of building applications or systems with objects. Implementing an Object-Oriented-Design can done using almost any programming language. However, OO languages like Smalltalk, C++, and Java provide mechanisms to make the implementation easier.

It is a natural evolution from earlier innovations to programming language design .It is more structured than previous attempts at structured programming; and it is more modular and abstract than previous attempts at data abstraction and detail hiding. Three main properties characterize an object-oriented programming language:

- ***Encapsulation:*** Combining a record with the procedures and functions that manipulate it to form a new data type: an object.
- ***Inheritance:*** Defining an object and then using it to build a hierarchy of descendant objects, with each descendant inheriting access to all its ancestors' code and data.
- ***Polymorphism:*** Giving an action one name that is shared up and down an object hierarchy, with each object in the hierarchy implementing the action in a way appropriate to itself. [2,6,9].

3. Reverse Engineering (RE)

Its starts with an existing system, analyzing the system to identify its components and interrelationships among components.

The principal benefit is recovery of useful information and structures. Reverse engineering (RE) is performed to gain a better understanding of an existing system. In other word reverse engineering is an analysis process. As a result, software engineering rebuilds design specification (improving the description of its architecture, its description) to set the stage for a forward engineering.

The core of RE is an activity called extract abstraction .The engineer must evaluates the old program and from source code, extract a meaningful specification of the processing that is performed, the user interface that is applied and the program data structure or database that is used [4,5,6].

4. Extensible Markup Language XML

Extensible Markup Language (XML) is a data storage toolkit; a configurable vehicle for any kind of information, an evolving and open standard embraced by everyone from bankers to webmasters. It’s a protocol for containing and managing information.

It is a subset of the Standard Generalized Markup Language (SGML) SGML is a way to express structure and content in different types of electronic documents, it has been around for more than a decade.

The structure in XML is built up by markup tags and character data in-groups that are normally called elements. Each element represents a logic component of the XML document. All elements consist of one of the following constructs:

- A tag together with the character data that the tag describes.
- A tag together with other elements (which in turn can consist of other tags and elements and so on)

Example:

```
<Owner>
<Name> Kalle Karlsson </Name>
<Address> G◌tgatan 1 </Address<
<Zip code> 11111 </Zip code>
<City> Stockholm </City>
</Owner>
```

The XML document was created into editor and the processor read this document and convert it into tree of elements and this processor pass this tree to viewer that display it like Internet Explorer or Netscape this is life cycle of XML document this is shown in Figure (1).

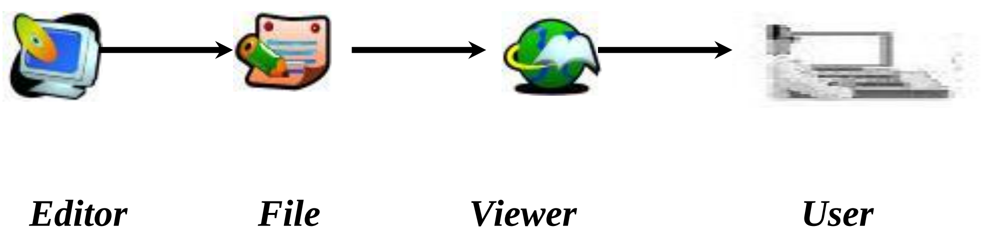


Figure (1) The XML Document Life Cycle.

The values stored within the elements can basically be of three different types, parsed character data (PCDATA), unparsed character data (CDATA), or processing instructions (PI).

- **PCDATA: -** Is used to store marked-up text that will be evaluated by the XML parser.
- **CDATA: -** Is used to store marked-up text so that the markup is not evaluated, i.e. CDATA sections provide a way of making sure specific markup is not interpreted as markup. CDATA sections begin with the string <![CDATA, and end with the string]]>.
- **PI: -** Holds processing directions and information passed to XML parsers and programs. PI instructions always start with <?, and end with ?>. An example of a PI is the code at the beginning of every XML document that tells the parser which version of XML the document contains <? XML version = “1.0”? >[1,3].

<u>4.1 XML Data</u>

In the XML specification, the contents of the tagged elements are always interpreted as a string. This is not satisfactory for all purposes since many applications need to be able to specify rigid constraints on the data they handle. They need to know if a certain data is an integer, float or a string. Some applications also need to specify within what range a certain value is allowed to be. The typical example is databases, which have very stringent constraints on their field-values

The XML-Data specification provides a standardized way to describe datatypes, ranges, default values, and other information concerning XML elements. In the specification, the datatype of an element is defined using a standardized datatype-namespace and a specific datatype attribute. Together this construct is referred to as the dt:dt attribute (the first dt is for the datatype namespace and the second dt names the datatype-attribute)[3,10].

4.2 XML Document

An XML document can be defined as a linear series of characters and references to other objects. An XML processor starts at the beginning of the document and works down to the end. XML provides a mechanism for allowing the text and objects in the document to be organized non-linearly. The parser then reorganizes it to the linear structure. The mechanisms that make this possible are called entities. An entity can be as small as a single character or as large as an entire XML document. An XML document can be broken up into many files on a hard disk or objects in a database and each of them is called an entity in XML terminology. Entities can even be spread across the Internet. Whereas XML elements describe the XML documents logical structure, entities keep track of the location of the chunks of bytes that make up the document.

Very simplified an entity consists of a name and a content. The content is the actual stored data and the name is used to refer to that data. There are several different kinds of entities used for different purposes. If an entity is defined without any separate storage file, and the content is given in its declaration, the entity is called an *internal entity*. All internal entities are parsed entities. This means that the XML processor parses them like any other XML text. The name parsed is somewhat badly chosen since the entities are in fact unparsed until they are actually used [7,8,10].

- **Internal entities used as an abbreviation**

  <! ENTITY dtd “document type definition”>

- **Internal entities with markup**

  <! ENTITY dtd “<term>document type definition</term>”>

4.2.1 The XML Schema

A schema defines the content of a number of XML-documents
It defines which elements and attributes can be included: -

1. The element content.
2. The order of elements.

Schema substitutes DTD Think of classes (schema) and instances (document) Schema is saved with postfix(* .xsd) A document is validated against a schema. A schema is an XML-document [10].

4.3 XML Document Type Declaration (DTD)

An important part of XML is the ability to at the beginning of an XML document store information about what the rest of the document will contain; this information is called the Document Type Declaration (DTD).

The DTD defines what markup tags can be used in the document, what order they can appear in, what other tags they can contain and so on. In XML it isn't strictly necessary to have a DTD associated with each XML document but it is considered bad manners not to have one. Also a DTD is necessary to be able to check if an XML document is valid and/or well structured. These two terms will be defined below. The following example shows the DTD for the previous employee example.

The DTD consists of a number of <! ELEMENT> and <! ATTLIST> tags that are used to define the structure of the rest of the XML document. This is the DTD for the previous employee example.

```
<DOCTYPE Employees [
<!ELEMENT Employees (Person)>
<!ELEMENT Person(Name)>
<!ATTLIST Person
email PCDATA #IMPLIED
phone PCDATA #IMPLIED
fax PCDATA #IMPLIED>
<!ELEMENT Name #PCDATA>
]>
```

The DTD is used to define the model that the rest of the data in the document must follow. The DTD can be fetched from an external source or be imbedded in the XML document itself. Today there exist big efforts to develop standardized DTDs for different areas [1,7,8].

4.4 Document as a Tree

An XML document is a tree, with each edge in the tree representing element nesting (or containment). To enable efficient processing on large databases, we require set-at-a-time processing of data. In other words, we require a bulk algebra that can manipulate sets of trees: each operator on this algebra would take one or more sets of trees as input and produce a set of trees as output. Using relational algebra as a guide, we can attempt to develop a suite of operators suited to manipulating trees instead of tuples. We have devised such algebra, called Tree Algebra XML(TAX)[10].

XML needs to refer to the information in a well-formed XML document.

An XML-document is well formed when: -

1. It only contains one top-level element; this element name must be unique (often-called root element).
2. Tags are properly balanced.
3. Attribute names are unique and their values quoted.

Basically, the requirement for well formedness ensures that serialized XML documents can be transformed into labeled trees. Every XML document therefore consists of elements hanging together in a logical tree structure. There is always one element that contains all the other elements; this element is called the root element. The tree structure of the elements is a vital part of XML and is used when an application wants to read and interpret an XML document [3,8].

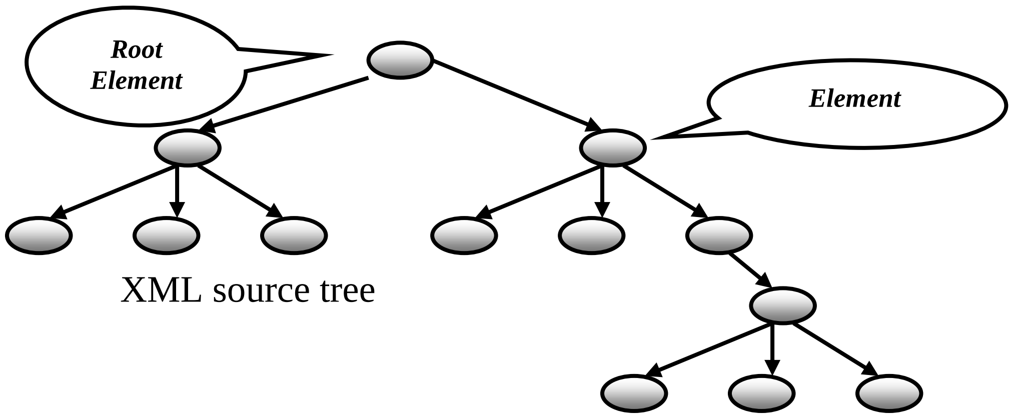


Figure 2: The structure of the XML source tree; the root node and the connected elements.

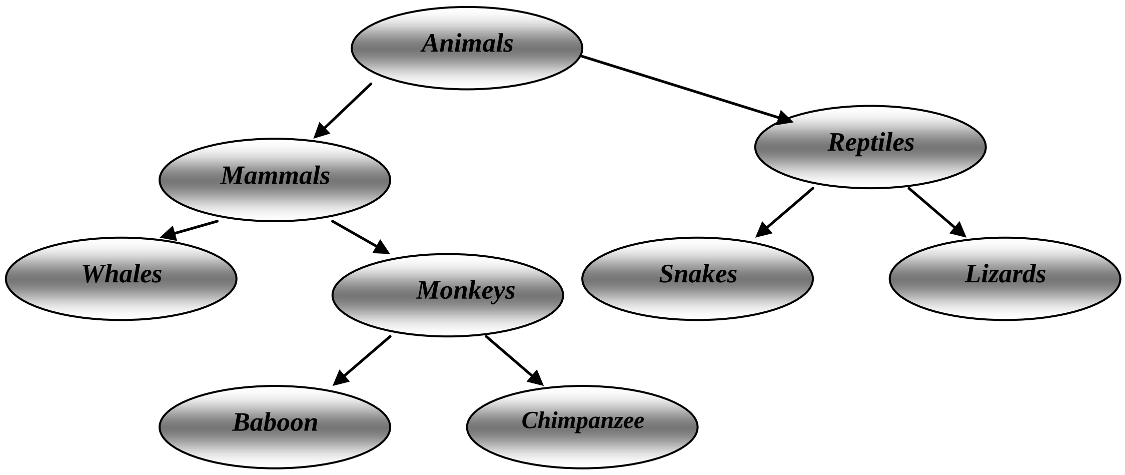


Figure 3: Shows a graphical representation of the binary tree (containing animals).

```
<?XML version = “1.0” ?>
<!DOCTYPE BINARY_TREE [
<!ELEMENT BINARY_TREE (ROOT)>
<!ELEMENT ROOT (VALUE,CHILDREN)>
<!ELEMENT CHILDREN(LEFT_CHILD?,RIGHT_CHILD?)>
<!ELEMENT LEFT_CHILD(VALUE?, CHILDREN?)>
<!ELEMENT RIGHT_CHILD(VALUE?, CHILDREN?)>
<!ELEMENT VALUE(#PCDATA)>]>
<BINARY_TREE>
<ROOT>
<VALUE>Animals</VALUE>
<CHILDREN>
```

```
<LEFT_CHILD>
<VALUE>Mammals</VALUE>
<CHILDREN>
<LEFT_CHILD>
<VALUE>Whales</VALUE>
</LEFT_CHILD>
<RIGHT_CHILD>
<VALUE>Monkeys</VALUE>
<CHILDREN>
<LEFT_CHILD>
<VALUE>
  Baboon
</VALUE>
</LEFT_CHILD>
<RIGHT_CHILD>
<VALUE>
Chimpanzee
</VALUE>
</RIGHT_CHILD> </CHILDREN>
</RIGHT_CHILD>
</CHILDREN>
</LEFT_CHILD>
<RIGHT_CHILD>
<VALUE>Reptiles</VALUE>
<CHILDREN>
<LEFT_CHILD>
<VALUE>Snakes</VALUE>
</LEFT_CHILD>
<RIGHT_CHILD>
<VALUE>Lizards</VALUE>
</RIGHT_CHILD>
</CHILDREN>
</RIGHT_CHILD>
</CHILDREN>
</ROOT>
</BINARY_TREE>
```

4.5 Native XML Databases

The most recent advance in database technologies regarding XML is Native XML databases. Native XML databases (NXDs) are designed specifically for storing XML documents. Like other databases, they support features such as concurrency control, scalability in multi-user environments, data integrity, transaction control, security, query languages, etc. The difference is that their internal model is specifically designed for persisting XML.

NXDs have the XML document as their fundamental unit. This means that document order, processing instructions, comments, etc are preserved in opposition to XEDs. For the same reason the existence of DTDs and XML [1,3,7].

4.6 XML Parser

An XML parser is software solely dedicated to reading and interpreting XML documents to enable further processing by other applications. They do this by giving the developer access to classes that can process XML documents. These classes can then be called by other classes in an application through a common interface (almost all parsers support the DOM Level 1 interface). The most popular language to write XML parsers in is Java even though there are also parsers available in other languages.

Two pairs of traits distinguish the different XML Parsers:

- Whether they are *validating* (checks DTD) or *non-validating* (checks for well-formedness, no DTD checking).
- Whether they are lightweight and therefore intended for use in applets or whether they are best suited for full-fledged applications [1,3].

4.7 Programming with XML

The Extensible Stylesheet Language (XSL) was still under development and subject to change. (XSL) which is a programming language designed to transform one XML document into other documents (that may be other XML documents). The popular approach to programming with XML is using the Extensible Stylesheet Language (XSL).

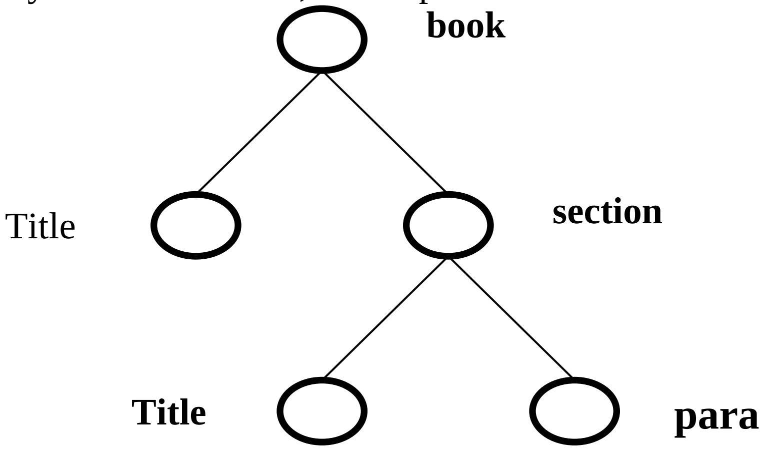
XSL is made up of two parts:

1. XSL Transformations (XSLT)

2. XSL Formatting Object (XSL-FO)

XSLT is a stylesheet language for defining transformations of XML documents into other XML documents. XSLT-FO is a language for specifying low-level formatting of XML documents, and is not covered here. XSLT is an XML language, i.e. a given XSLT stylesheet (program) is an XML document. XSLT is a declarative language; you state what you want, but not how you want it done. XSLT uses pattern matching and template rules to perform transformations, as illustrated in example bellow Template rules contain rules to be applied when specified nodes are matched. Template rules identify the nodes to which they apply (they match) by using a pattern, e.g. the template rule <xsl: template match ="/"> will match the root node of any XML document, as the pattern ”/” match the root node [7,8,10].

Example: -



```
<book>
<title> chicken soup </title>
<section>
        <title> Introduction </title>
        <para> I’ve always.. <para>
</section >
</book>
```

Template rule for book

```
<xsl:template match=”book”>
<body>
<h1> <xsl:value-of select=”title”/> </h1>
</body>
</xsl:template>
```

Template rule for section

```
<xsl:template match=”section”>
        <h2><xsl:value-of select=”title”/></h2>
        <xsl:apply-template select=”para”/>
</xsl:template>
```

5. The proposed System

The proposed system is designed to **c**onvert **o**bject-**o**riented - **p**rogramming (OOP) into **XML** schema and Document Type Declaration (DTD) files "COOPXML", it is used for OOP with (*.java; *.class) extension .

This system is accepting OOP with (*.java ; *.class) extension and using Reverse Engineering to analysis input OOP system to extract the information about system components such as classes ,objects, relationships, and the structure of input system . After this point the RE generate document data that document input system. The proposed system using Java XML (JXML) parser to read Document data that result from RE step and analysis it to generate Tree Algebra XML (TAX) and finally we used XSMLT to translate TAX into XML files (*.xmi) and DTD associative files.

The general algorithm of the proposed system is: -

Algorithm Conversation

Input: OOP system with (*.java;*.class)extension

Output: XML files (*.xmi), Document Type Declaration (DTD)

```
Step 1:Begin
Step 2: Analysis OOP to extract its components of input system using
        parsing levels.
Step 3: Generate data document (DD) from parsing level.
Step4: Using Java XML (JXML) parser to analysis DD patterns and
        generates Database.
```

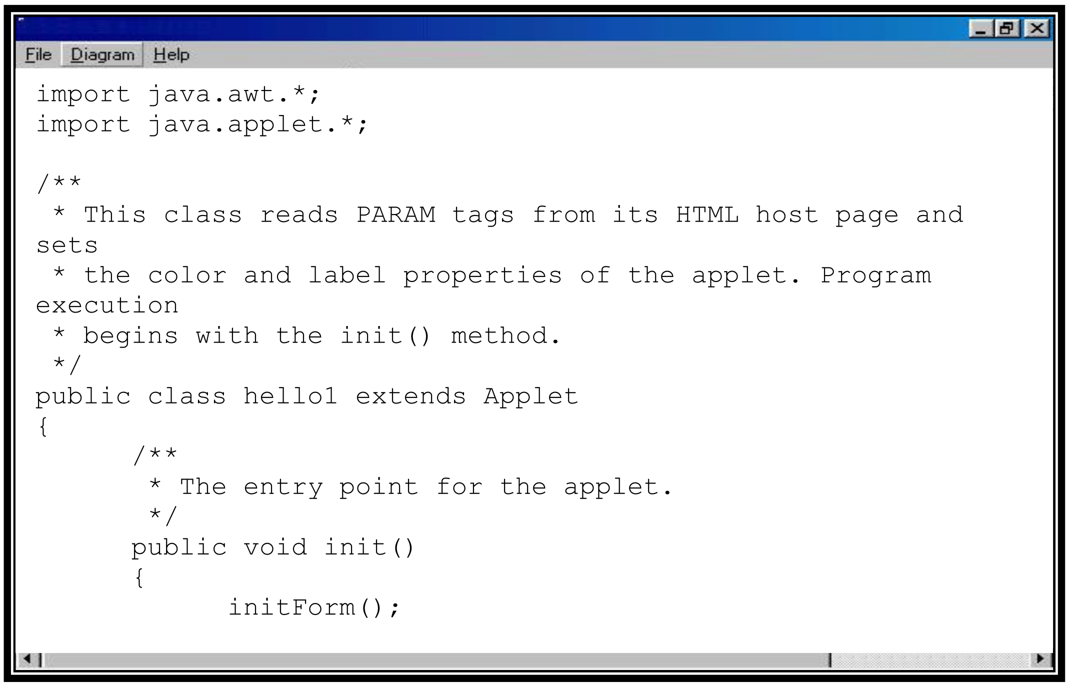
Step5: Parse the entire database (XML document) and build a TAX tree.

Step 6: Build XML based description XML tree (TAX) using XSLT to translation TAX into XML schema (*.xmi) and associative Document Type Declaration (DTD) files.

Step 7: end.

The Practical Example: -

The input OOP with Java extension is shown in Figure (4).

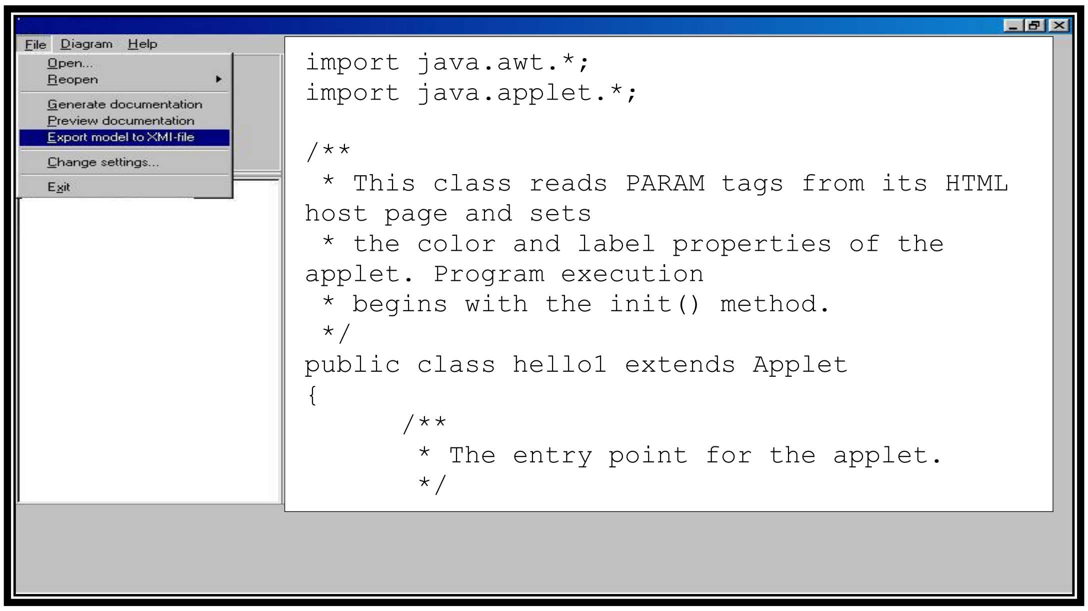


```
import java.awt.*;
import java.applet.*;

/**
 * This class reads PARAM tags from its HTML host page and
sets
 * the color and label properties of the applet. Program
execution
 * begins with the init() method.
 */
public class hello1 extends Applet
{
      /**
       * The entry point for the applet.
       */
      public void init()
      {
            initForm();
```

Figure (4) Accept OOP system.

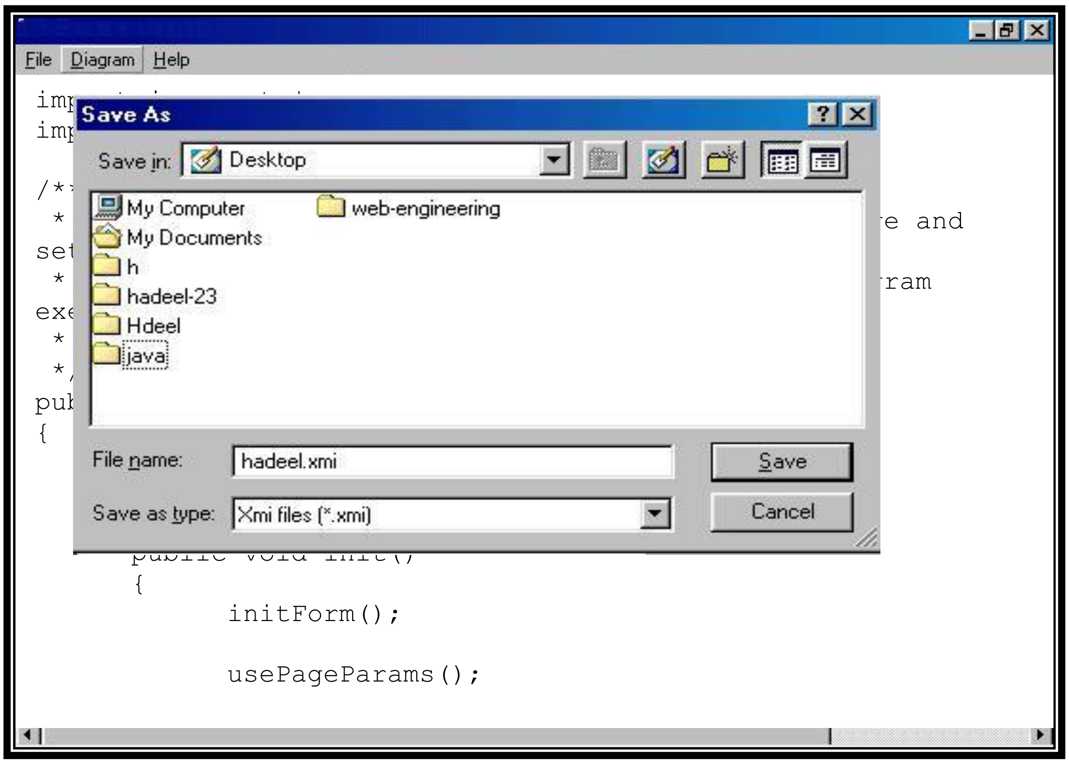
The proposed system in this point start the conversation operation and it must specify the name of output file (*.xmi) that save the conversation results, this is shown in figures (5,6)



```
import java.awt.*;
import java.applet.*;

/**
 * This class reads PARAM tags from its HTML
host page and sets
 * the color and label properties of the
applet. Program execution
 * begins with the init() method.
 */
public class hello1 extends Applet
{
	/**
	 * The entry point for the applet.
	 */
```

Figure (5) Select Convert Operation.



Save in: Desktop

File name: hadeel.xmi

Save as type: Xmi files (*.xmi)

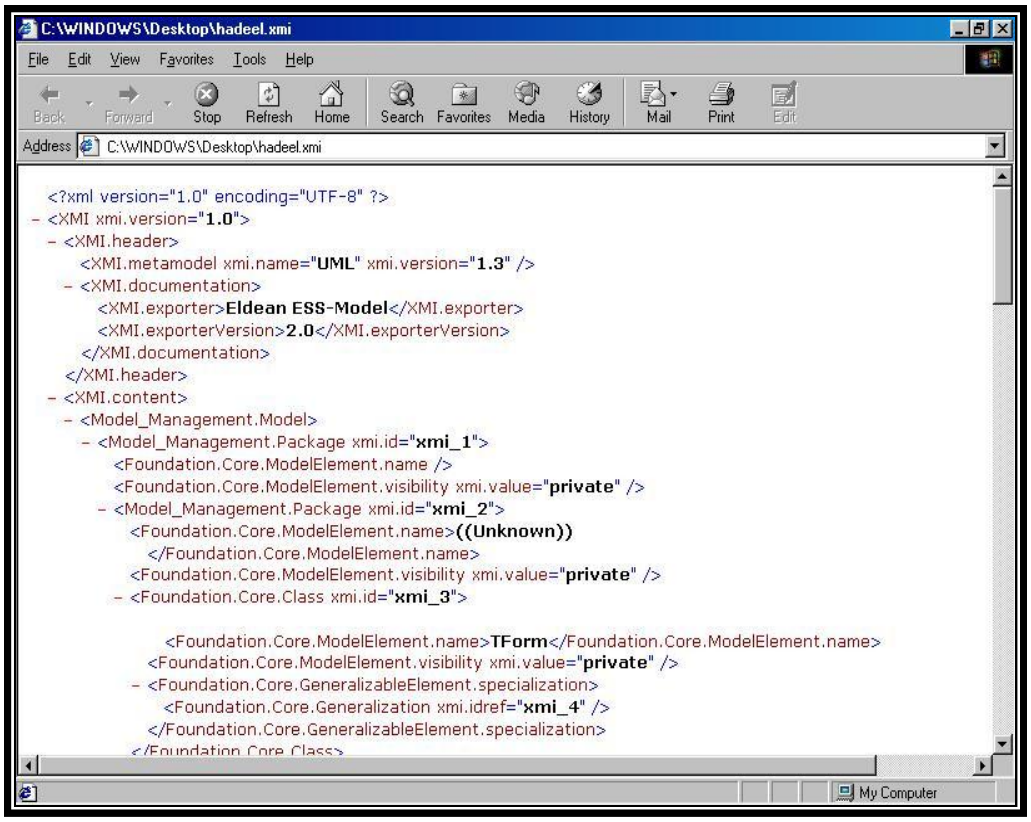
```
imp
imp

/**
 *                                        e and
set
 *                                        ram
exe
 *
 */
pub
{
	public void init()
	{
		initForm();

		usePageParams();
```

Figure (6) Create Folder.

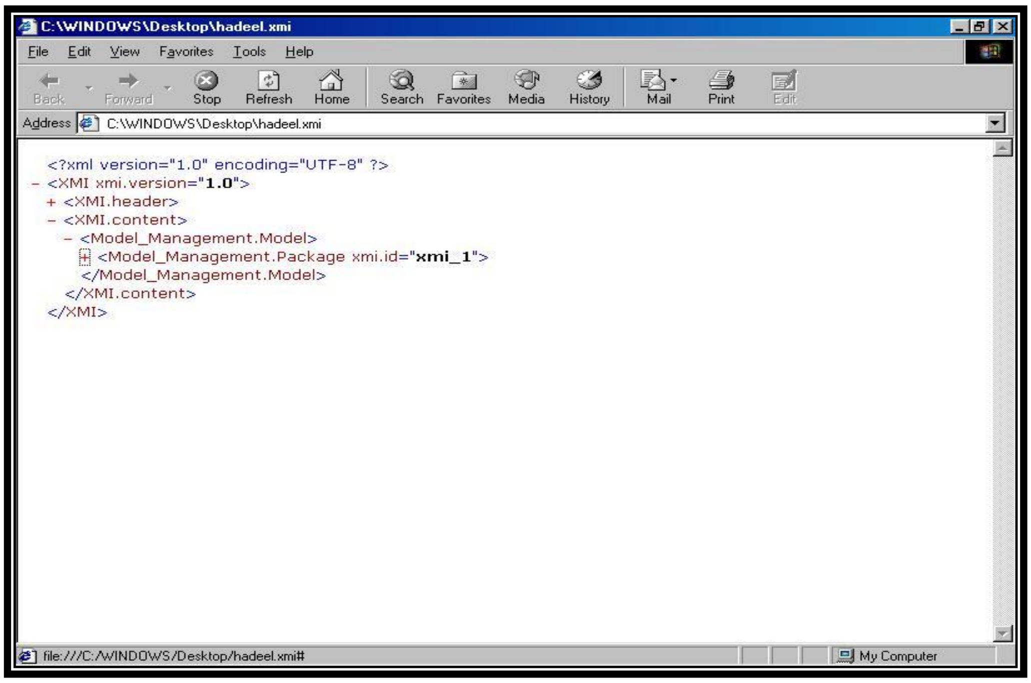
The output file with (*.xmi) was display into figures (7,8)


C:\WINDOWS\Desktop\hadeel.xmi
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit
Address C:\WINDOWS\Desktop\hadeel.xmi
My Computer

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <XMI xmi.version="1.0">
  - <XMI.header>
      <XMI.metamodel xmi.name="UML" xmi.version="1.3" />
    - <XMI.documentation>
        <XMI.exporter>Eldean ESS-Model</XMI.exporter>
        <XMI.exporterVersion>2.0</XMI.exporterVersion>
      </XMI.documentation>
    </XMI.header>
  - <XMI.content>
    - <Model_Management.Model>
      - <Model_Management.Package xmi.id="xmi_1">
          <Foundation.Core.ModelElement.name />
          <Foundation.Core.ModelElement.visibility xmi.value="private" />
        - <Model_Management.Package xmi.id="xmi_2">
            <Foundation.Core.ModelElement.name>((Unknown))
              </Foundation.Core.ModelElement.name>
            <Foundation.Core.ModelElement.visibility xmi.value="private" />
          - <Foundation.Core.Class xmi.id="xmi_3">

                <Foundation.Core.ModelElement.name>TForm</Foundation.Core.ModelElement.name>
              <Foundation.Core.ModelElement.visibility xmi.value="private" />
            - <Foundation.Core.GeneralizableElement.specialization>
                <Foundation.Core.Generalization xmi.idref="xmi_4" />
              </Foundation.Core.GeneralizableElement.specialization>
            </Foundation.Core.Class>
```

Figure (7) XML Formula for Input system.


C:\WINDOWS\Desktop\hadeel.xmi
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print Edit
Address C:\WINDOWS\Desktop\hadeel.xmi
file:///C:/WINDOWS/Desktop/hadeel.xmi#
My Computer

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <XMI xmi.version="1.0">
  + <XMI.header>
  - <XMI.content>
    - <Model_Management.Model>
      + <Model_Management.Package xmi.id="xmi_1">
      </Model_Management.Model>
    </XMI.content>
  </XMI>
```

Figure (8) The OOP system display into XML Environment.

6. Conclusion

Through this paper and use of this system, we came out with few points that summarize our conclusions, these are: -

1. XML is designed to store, transmit and exchange data, XML can encode the representation for :
   - An ordinary document
   - A structured (database) record, such as an appointment record
   - An object, with data and methods (e.g. a Java object or an ActiveX control).
   - Graphical presentation (such as an application’s user interface).
2. In this paper we could produce an XML-document which is a well-formed document.
3. It is a flexible way to create “ self-describing data”, and share both the format and data on the World Wide Web, Internets.
4. This work helps developer to great the optimal Document Type Declaration (DTD) about input Object-Oriented System.
5. The XML uses different XML elements for each composition operations. An XML solution to composition language problem by describing how it addresses each of requirements listed earlier.
6. This work can be used to represent OOP as Database documents into textual way.
7. This work can be used as a tool for supporting Web-based applications.
8. This work can be using from designer as a tool for semantic web (as a translation tools).

7. References

1. Bray T., Paoli J., and C.M. Sperberg-McQueen (eds.): Extensible Markup Language (XML) 1.0,W3C Recommendation 10-February-1998, HYPERLINK: -http://www.w3.org/TR/REC-xml.
2. Davis S.R.," Programming Visual J++ 6.0", Micro software sires, 1999.
3. Erik, T. Ray, " Learning XML", Oreilly & associates, Inc., 2001.
4. Fisher A.S.," CASE using Software development tools ", John Wiley, 1998.
5. Hamlet D., Maybee J.," The Engineering of software ", Addison-Wesley longman, 2001.
6. Jacobson I.," Object-Oriented Software Engineering - A Use Case Driven Approach", Addison-Wesley, 1992.
7. James Clark (ed): XSL Transformations (XSLT), W3C Recommendation 16 November 1999 ,http: //www.w3.or g/T R/xslt
8. James Davidson: Java API for XML Parsing, Version 1.0 Final Release, HYPERLINK:- http://java.sun.com/aboutJava/community.
9. Morgan B.," Visual J++ Unleashed", Second Edition, 2002.
10. Thompson H. S., Beech D., Maloney M., Mendelsohn N. (eds.): XML Schema Part 1:Structures W3C Working Draft 7 April 2000, HYPERLINK :-http://www.w3.org/TR/2000 / WD-xmlschema.