International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 19 (2017) pp. 8632-8641

Solving the Interoperability problem between UML modeling tools: Modelio and ArgoUML

W. Lahlayl[1], N. Berbiche[2], F. Guerouate[3] and M.Sbihi [4]

[1]PhD student at LASTIMI laboratory Mohammed V University in Rabat, Morocco.
[2,3,4] Research Professor at LASTIMI laboratory and Professor in the Department of Computer Sciences, Higher School of Technology - Sale, Mohammed V University in Rabat, Morocco.

[1]Orcid ID: 0000-0003-4545-8464

Abstract

Model-driven engineering is an approach that applies to software engineering and advocates the use of models throughout the development cycle. The architecture of computer systems is presented as models described by many UML modeling softwares. The serialization of these models is done by another standard published by the OMG called XML Metadata Interchange (XMI) which is recommended to ensure interoperability between UML modeling tools. However, the independent evolution of these standards, namely UML and XMI, has led to interoperability problems between UML modeling software. It is within this context that we present our approach, whose objective is to identify and solve the problem of interoperability in the XMI files of the UML language's class diagrams and mainly between the Modelio and ArgoUML tools. The use of XSLT transformation rules in the context of the template approach of model transformations has allowed the reusability and interoperability of class diagrams between the two modeling tools.

Keywords: MDA, Interoperability, UML, XMI, XSLT, Modelio, ArgoUML, Class diagram.

Nomenclature

UML	Unified Modeling Language
MDA	Modeling Driven Architecture
M2M	Model to Model
M2T	Model to Text
EMF	Eclipse Modeling Framework
XMI	XML Metadata Interchange
XSLT	eXtensible Stylesheet Language Transformations
XSL	eXtensible Stylesheet Language
DTD	Document type definition
XSD	XML Schema Definition

INTRODUCTION

Information systems are an essential element in the organization of our enterprises and, more generally, of our society. Today, they are built on the aggregation of computer systems that must be maintained and evolved with flexibility and without difficulty. The development of these computer systems follows several steps, among which are design and modeling that consist in presenting the architecture of the computer softwares as models.

This approach is called Model Driven Architecture (MDA). The preferred language to describe MDA is UML. UML modeling consists of many design diagrams that can be modelled by several tools such as Modelio, ArgoUML, Bouml etc. The storage and exchange of UML models are made by a standard published by the OMG (Object Management Group, http://www.omg.org/): XML Metadata Interchange (XMI) which was developed with the goal of allowing the exchange of models between the various UML modeling tools by formulating them in an interoperable format. However, due to the separate evolution of UML and XMI standards and the evolution of languages that define the structure of XML documents, UML tools produce non-interoperable models. This interoperability issue eliminates the reusability and scalability of design diagrams.

In this article, we have chosen to address the problem of interoperability between two modeling tools UML, Modelio and ArgoUML. We propose an approach to identify and solve the interoperability problem at the level of the XMI files resulting from the UML class diagrams produced by Modelio and ArgoUML using the transformation approach by Template XSLT.

In Section 2, we will present a state of the art where we will give an overview on model transformation approaches in the MDA framework and mainly the XSLT template Approach. Then in Section 3 we will present the motivations that has led us to do this work. In Section 4, we will discuss the literature review that has been done in the field of interoperability. Section 5 concerns the approach used, it presents the XSLT

transformation rules performed on the XML files of the UML tools and the methodology followed. And finally, Section 6 presents the results of our approach and our vision for future work.

STATE OF THE ART

Model transformation approaches

In the MDA context, there are different approaches to transforming models, each one providing a specific way for the processing of transformation rules. The approaches can be divided into two types of transformation: the Model to Text (M2T) transformation and the Model to Model (M2M) transformation.

- M2T (Model to Text) transformations: transformations aim for generating code or documentation. The M2T transformations are the subject of the MOFM2T project, which is part of the MDA project [1].

- M2M (Model to Model) transformations: These are transformations from models to models, these transformations concern all the tasks to be executed in order to create a model that respects the technical specifications of the target environment from another model. The technological platform that technically represents this type of transformation in the MDA approach is the MOF 2.0 QVT standard.

The following figure presents an overview of the MDA transformation.

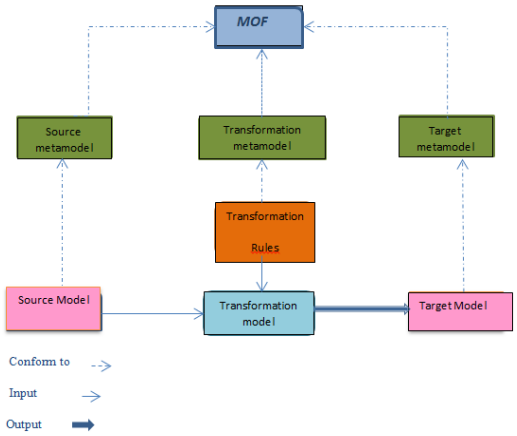


Figure 1: Overview of MDA transformation

There are different approaches to execute a transformation of M2M models, each offering a very precise way for the elaboration of the transformation rules.

✓ The programming approach:

This approach is based on the use of programming languages to describe the transformation rules in the form of computer programs, so the model transformation is a computer application that manages models. This approach is considered as a direct manipulation approach [2]. The programming approach has two different visions:

1) The imperative approach: Similar to imperative programming, this approach relies on the foundations of structured programming and defines the transformations in the form of sequences of instructions contained in functions and procedures to change the state of the model at runtime. The interest of such an approach lies in the fact that it remains relatively familiar to developers, the syntax and the semantics it uses is close to the Object Constraint Language (OCL) [3].

2) The API Approach: Some APIs (Application Programming Interface) of transformation are described in imperative programming languages and are then provided as libraries that allow the description of the transformation process according to the syntax of the language used. The results are then of considerable performance. However, the developer is responsible for organizing and describing all the steps explicitly in terms of mandatory declarations [3].

✓ The modeling approach:

Also called the transformation by rule approach, this approach consists in modeling the transformations themselves in order to make them perennial and productive by applying model engineering. It can be realized by one of the following two approaches:

1) The graph approach: This is a mathematical formalism that applies graph theory to transform models, considering them as graphs. The transformation strategy in this approach consists of a substitution and a mapping between the source and target model, which uses graph rule syntax to take a Left Hand Side Graph (LHSG) and transform it into an RHSG (Right Hand Side Graph) [19]. The complexity of this approach lies in the non-deterministic aspect of the strategy of application in the transformations rules [2], which implies that solutions based on this paradigm are not used a lot in concrete terms.

2) The Declarative Approach: In this approach, a rule maps a set of concepts invoked in the source model and a set of concepts that should be adopted in the target model. The implementation is carried out by an inference engine. However, one of the main disadvantages of this approach is the high workload that the developer needs to specify, which is tedious [4].

✓ The hybrid approach:

This approach is the most recent among the other transformation approaches, combining the declarative and imperative approach. The declarative approach is generally used for the definition and selection of transformations that can be applied, while the imperative approach is well adapted to the description of the transformation strategy [3].

✓ The Template Approach:

It consists in defining patterns that are parametrized for the target models. The parameters are provided by the values contained in the source models in order to be able to mount the output model. The reference implementation representing this approach is the XSLT approach that implements the XSLT language (eXtensible Stylesheet Language Transformations). This language was originally designed for transforming XML documents (eXtensible Markup Language) into other formats. In the context of MDA, the models are serialized in XMI format (XML Metadata Interchange), which is a variant of the XML language, this explains why such an approach adapts perfectly to model transformations in such a context. Nevertheless, this approach has disadvantages of performance and efficiency once the model to be transformed reaches a certain level of complexity.

Figure 2 shows the hierarchy of the M2M transformation.

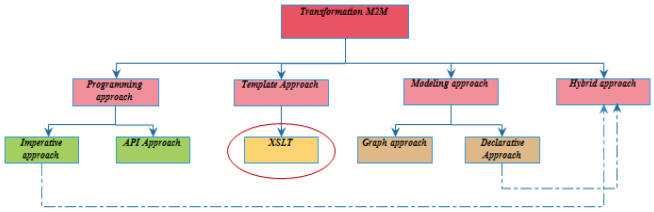


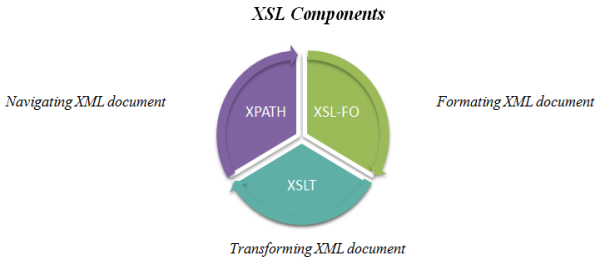
Figure 2: Different approaches of M2M model transformations

Our work consists in carrying out two transformations of models (of the M2M type): Firstly in order to adapt the structure of the models obtained by the serialization process from the ArgoUML tool to the format recommended by Modelio, also to carry out a transformation in the inverse path, namely to adapt the models obtained by Modelio to the format used by ArgoUML. To achieve this goal, we are interested in the Template approach, which is based on the XSLT language (eXtensible Stylesheet Language Transformations). Our work handles the transformation of UML models, which are serialized in the XMI format (XML Metadata Interchange): a variant of the XML language. For this reason, the use of the XSLT language has proved to be judicious.

The XSLT transformation language

In order to solve the problem of interoperability of the UML models serialized in XMI files, it is obvious to think about the approaches of MDA transformation and more precisely the approach by XSLT Template. Since the problem is related to the XMI file which is a variant of the XML language.

The XSL Transformations (XSLT) is a language used to apply transformation rules on an XML document in order to obtain another XML document. These transformation rules are described in XSL stylesheets using XPATH path requests.



- XSLT transforms the XML document into another XML document that can be readable and well structured.
- XPATH is used to navigate the source document in order to apply XSLT transformations into the tags found.
- Once the transformations are applied, XSL-FO formats the document so that it is well structured.

UML: Class Diagram

UML [5] is a graphical modeling language that is widely used by systems based on object-oriented programming. It is a language based on the OMG's MOF specifications. UML provides a set of diagrams for modeling and designing the systems to be developed. UML version 2.5 defines two types of UML diagrams: structure diagrams and behavior diagrams.

- Structure diagrams show the static structure of the system and its parts on different levels of abstraction.

- Behavior diagrams show the dynamic behavior of objects in a system, which can be described as a series of changes to the system over time.

XMI (XML Metadata Interchange)

UML models are serialized in XMI (XML Metadata Interchange) files [6]. It has also been proposed by the OMG as a standard for the interoperability and serialization of UML models. The XMI language is based on XML syntax.

MOTIVATIONS

The first step into this work is to define the interoperability problems between UML tools. For this reason, we carried out a comparative study of the different UML modeling tools. This study has allowed us to identify the factors contributing in the interoperability problems of UML tools.

This study was performed on: ArgoUML[7], Bouml [8], Entreprise Architect [9], Visual paradigm[10], Modelio[11], Papyrus[12], Topcased[13], PowerAMC[14], Eclipse UML free edition [15] and Magic draw[16].

The table 1 shows the XMI version of import and export of each tool and the UML version.

We noted that there is a correspondence between XMI and UML. Returning to the specifications of the OMG, we could detail all the different existing versions of UML and XMI. The colored boxes indicate that there is no match between UML and XMI. The DTD and XSD annotations are validation files for XMI files that represent the OMG specifications.

According to the table 2, there is no unique version of XMI for a given version of UML.

- ✓ UML 1.3 XMI 1.0
- ✓ UML 1.3 XMI 1.1
- ✓ UML 1.4 XMI 1.1
- ✓ UML 2.1 XMI 2.1
- ✓ UML 2.4 XMI 2.4
- ✓ UML 2.4.1 XMI 2.4.1
- ✓ UML 2.4.1 XMI 2.4.2
- ✓ UML 2.5 XMI 2.5.1

transformation approach by XSLT Template.

After reviewing the versions used for the import and export of each tool, we noted that there is a link between the versions of both XMI and UML. Going back to the OMG specifications, we were able to detail all the different versions of UML and XMI. On the other hand, the analysis of the equivalence of the UML versions allowed us to deduce that for the same XMI version, the way of describing the objects varies from one tool to another. For example ArgoUml and Bouml have the same XMI version and different versions of UML, the syntax of the tags are different. The table 3 below shows an overview of this finding:

All these observations led us to identify that the factors causing the interoperability problem of the UML tools are due to the numerous non-conformities writings of the tags in the XMI files.

Since the interoperability problem of UML tools is placed at level of the XMI file tags, the solution we have recommended to solve this problem is based on the use of the template

Table 1: Tested Modeling Tools

Tool	XMI Import version	XMI export version	UML version	Version Used	Current Version	Category
Visual paradigm	1.0 1.2 2.1	1.0 1.2 2.1	2.0	Version 12.0	Version 14	proprietary
Argouml	1.0 1.1 1.2	1.2	1.4	Version 0.34	Version 0.34	free
Enterprise Architect	1.1 2.1	1.1 2.1 2.5	1.3 2.1	Version 13	Version 13	proprietary
Modelio	2.1	2.1	2.1.1 2.2 2.3 2.4.1	Version 3.6.0	Version 3.6.0	free
PowerAMC	2.1 1.1 1.0	2.1 1.1 1.0	2.1 1.3 1.3	Version 15.1	Version 16.5.0	proprietary
PAPYRUS		2.1	3.0	Version 1.0.0	Version 1.1.0	free
BOUML	2.1	1.2 2.1	2.3	Version 6.9.2	Version 6.8.6	free
Topcased		2.1	3.0	Version 5.3.1	Version 5.3.1	free
Eclipse UML free edition	2.1	2.1	2.0	Version 2.1	Version 3.7.1	free
Magicdraw	2.1	2.1	2.5	Version 18.5	Version18.5	proprietary

Table 2: Different versions of UML and XMI existing

XMI\UML	1.3	1.4	1.5	2.0	2.1.1	2.1.2	2.2	2.3	2.4	2.4.1	2.5
1.0	DTD										
1.1	DTD	DTD	DTD								
1.2											
2.0											
2.1					XSD	XSD					
2.4									XSD		
2.4.1										XSD	
2.4.2										XSD	
2.5.1											XSD

Table 3: Different syntax of the XML file of each tool

UML tool	**ArgoUML**	**Bouml**
XMI File	<UML:Class xmi.id = '-64--88-1-4 2a32eb48:1527e16dc72:-8000:0000000000000962' name = 'Interface' visibility = 'public' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' isActive = 'false'/> <UML:GeneralizableElement.generalization> <UML:Generalization xmi.idref = '-64--88-1-4-2a32eb48:1527e16dc72:8000:00000000000009C9'/> </UML:GeneralizableElement.generalization> </UML:Class>	<UML:Class name="class" xmi.id="BOUML_0x2017d60" visibility="public" isAbstract="false" isActive="false"> </UML:Class> <UML:Generalization xmi.id="BOUML_0x9de5b8" isSpecification="false"> <UML:Generalization.child><UML:Class xmi.idref="BOUML_0x2017b80"/> </UML:Generalization.child> <UML:Generalization.parent> <UML:Class xmi.idref="BOUML_0x20180a8"/> </UML:Generalization.parent> </UML:Generalization>

RELATED WORKS

Interoperability problems [17] are not new and they constitute the research domains in which several communities, such as information systems, databases, software engineering, etc., are interested in. The problems of interoperability occur at different levels:

- Schematic: results from a different structuring and classification of information. It is closely related to design choices [17,18].
- Semantics: derives from the differences in the interpretation of information shared between different fields of application [17, 18].
- Syntactic: results from the use of different data models from one system to another. Different concepts are used to structure the same information (relations in the relational model, classes in the object model, XML tags, etc.) [1, 18].

In the field of computer systems, several studies have been made on interoperability problems and especially on semantic interoperability [19, 20, 21].

In the context of the syntactic interoperability which is the issue of our article, we can quote: Gaurav Bansal's work [22] in which he used the MDA programming transformation .He developed an XMI Parser which makes it possible to establish the exchange between Visual Paradigm and MagicDraw. However, the proposed solution does not address the inverse transformation of XMI files.

METHODOLOGY AND CONTRIBUTION

Evaluation and Comparison of XMI files introduced by Modelio and ArgoUML

To compare the XMI file (version 1.2 of XMI, version 1.4 of UML) generated by ArgoUML and the XMI file (version 2.1 of XMI and version 2.1.4 of UML) generated by Modelio, we created a class diagram containing all elements and the main relations of the class diagram as described in the following figure:

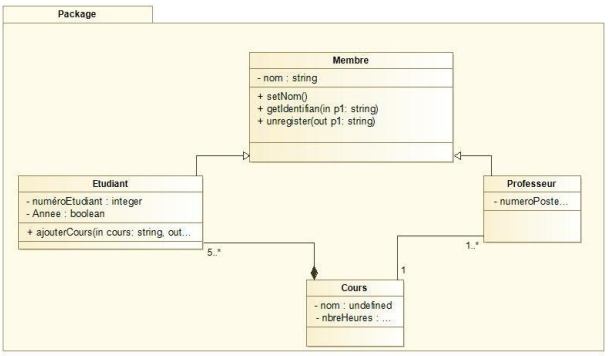


Figure 3: The used class diagram

When we imported the XMI file from the source tool into the destination tool (ArgoUML to Modelio and Modelio to ArgoUML) there was a loss of information, both tools could not mount the diagram, errors appear.

We also noted that Modelio and ArgoUML use different tags to represent the elements of the class diagram. For this reason, importing XMI files did not work.

For a better understanding of these differences, we compared the XMI files generated by ArgoUML and Modelio of the processed class diagram. Some differences are shown in the table 4. The elements appearing in Modelio but absent in ArgoUML are colored in red and those appearing in ArgoUML and absent in Modelio are colored in green.

By comparing the common tags of the XMI files generated by both tools, we noted that the method by which the information of each element of the class diagram is described changes from one XMI file to another. The table 5 shows these differences.

Table 4: The difference between ArgoUML and Modelio tags

Modelio	***ArgoUML***	***Observations***
Name of tag xmi:XMI ***Attributes :*** xmi:version xmlns:uml xmlns:xmi	***Name of tag*** XMI ***Attributes :*** xmi.version xmlns:UML timestamp	✓ The **xmi: XMI** and **XMI** tags are both: the root elements of Modelio and ArgoUML XMI files ✓ They have almost the same attributes except **xmlns: xmi** that does not appear in the XMI file of ArgoUml also **timestamp** does not appear in the XMI file of Modelio
Name of tag uml:Model ***Attributes :*** xmi:id name	***Name of tag*** uml:Model ***Attributes :*** xmi.id name isSpecification isRoot isLeaf isAbstract	✓ The **uml: Model** tag appears in both files. ✓ There are common attributes of this tag in both files such as **xmi: id** and **name**. ✓ In the XMI file of ArgoUML, this tag contains attributes other than the attributes mentioned above.
Name of tag eAnnotations ***Attributes :*** xmi:id source	***Name of tag*** XMI.documentation ***Attributes :*** None	✓ These two tags contain information about the XMI file. ✓ **eAnnotations** appears in the version 2.1 of XMI and **XMI.documentation** in version XMI 1.2
Name of tag contents ***Attributes : :*** xmi:type xmi:id name ="**exporterVersion**"	***Name of tag*** XMI.exporterVersion ***Attributes :*** None	✓ **contents** appears in version 2.1 of XMI and **XMI.exporterVersion** in the version 1.2. ✓ Although both files use two different XMI versions, **"exportVersion"** appears as the value of the **name** attribute of the **content** tag.

Table 5: Comparison of the tags that describe elements of the class diagram

Élément	Fichier XMI Modelio	Fichier XMI ArgoUML
Package	<packagedElement xmi:type="**uml:Package**" xmi:id=" " name="Package">	<**UML:Package** xmi.id=" " name="Package" >
Class	<packagedElement xmi:type="**uml:Class**" xmi:id="" name="Membre">	<**UML:Class** xmi.id=" " name="Membre">
Attribute	<**ownedAttribute** xmi:type="**uml:Property**" xmi:id=" " name="nom" visibility="private">	<**UML:Attribute** xmi.id=" " name="nom" visibility="private">
Method	<**ownedOperation** xmi:type="**uml:Operation**" xmi:id=" " name="setNom" visibility="public"/>	<**UML:Operation** xmi.id=" " name="setNom" visibility="public">
Association	<packagedElement xmi:type="**uml:Association**" xmi:id=" " >	<**UML:Association** xmi.id=" " name="">
Aggregation	<ownedAttribute xmi:type="uml:Property" xmi:id=" " name=" " visibility="public" type=" " **aggregation="shared"** association=" ">	<UML:AssociationEnd xmi.id=" " name="" visibility="public" **aggregation="aggregate"** >
Composition	<ownedAttribute xmi:type="uml:Property" xmi:id=" " name=" " visibility="public" type="" **aggregation="composite"** association="">	<UML:AssociationEnd xmi.id=" " name="" visibility="public" **aggregation="composite"** >
Generalization	<generalization xmi:type="**uml:Generalization**" xmi:id=" " **general**=" "/>	<**UML:Generalization** xmi.id=""> <UML:Generalization.child> <UML:Class xmi.idref=""/> </UML:Generalization.chil> <**UML:Generalization.parent**> <UML:Class xmi.idref=""/> </UML:Generalization.parent> </UML:Generalization>
Multiplicity	<**upperValue** xmi:type=" " xmi:id=" " value=""/> <**lowerValue** xmi:type="" xmi:id="" value=""/>	<UML:MultiplicityRange xmi.id=" " **lower**="" **upper**=""/>

To accomplish the XSLT transformation between these two files, it was necessary to represent the meta-model of the ArgoUML XMI file and the meta-model of the Modelio XMI file.

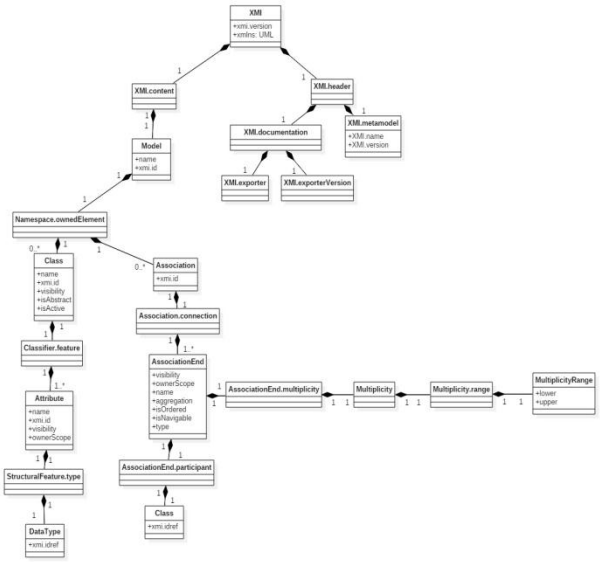


Figure 5: Metamodel of Modelio XMI file

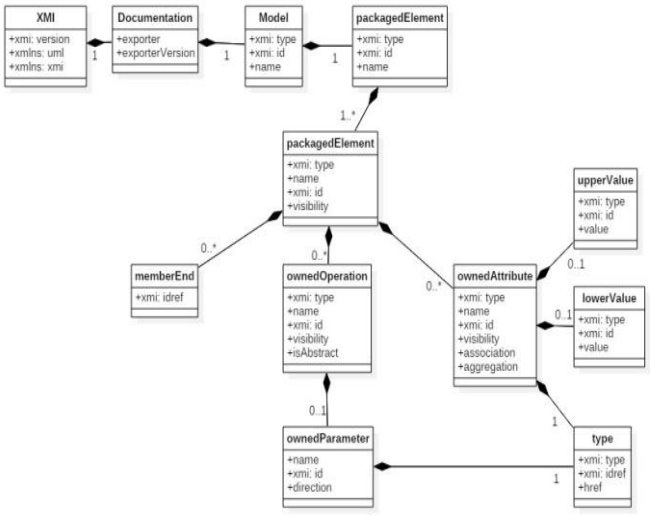


Figure 6: Metamodel of ArgoUML XMI file

Project Architecture

In our approach to solving the problem of interoperability between the Modelio and ArgoUML tools, we have developed

a tool that is based on three components:

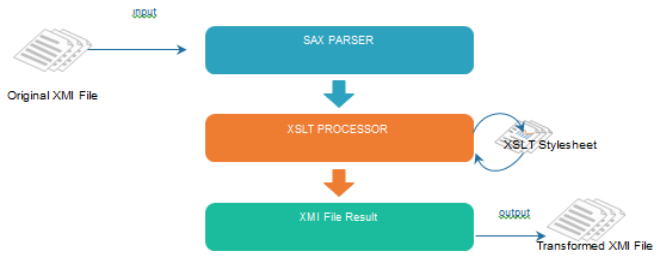


Figure. 7: The Architecture of the suggested solution

To extract the fields from the tags, we programmed a SAX type Parser. We chose this parser because it allows us to process very large XML files. This parser takes as input the source XMI file to be transformed and makes it possible to extract and visualize each tag and the attributes that constitute it. The following diagram shows the process of this Parser.

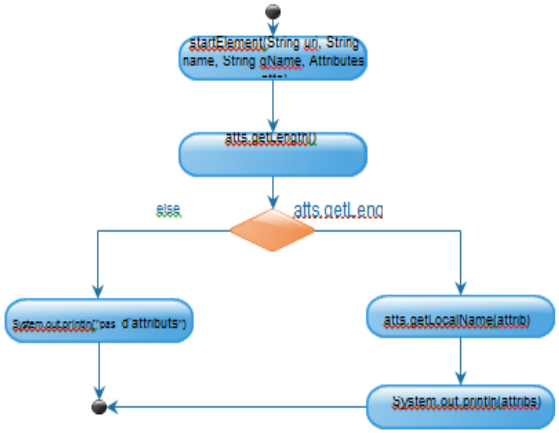


Figure 8: The parser process

The transformation of an XMI Modelio file to an ArgoUML XMI file and vice versa is done with an XSL style sheet using the XLST process (XPATH and XSLT) because the general structure of both files is based on the XML language. The transformation of the tags is accomplished with Templates that respect the correspondence between each tag of the target file and each tag of the source file using the XPath queries.

In this process, we imported the javax..xml.transform libraries in order to use them in programming the XSLT process. A part of the Java code used to transform the Source file is shown below:

```
TransformerFactory factory = TransformerFactory.newInstance();
        StreamSource xslStream = new StreamSource("transform.xsl");
        Transformer transformer = factory.newTransformer(xslStream);
        StreamSource in = new StreamSource("input.xmi");
        StreamResult out = new StreamResult(new File("output.xmi"));
```

Before the execution of the XSLT transformation rules, it is necessary to rename the XMI and UML namespaces of the target file to eliminate confusion with the XMI and UML namespaces of the source file. We also use "xmlns: uuid =" java: java.util.UUID "This is a namespace for defining random identifiers for the XMI: id field of each tag.

For Mod2Arg and Arg2Mod transformations, each element in the XMI file of the source tool's class diagram is transformed by a transformation rule and an XPATH request. For example, the transformation of the class Member and its attributes from the Modelio class diagram to Argouml (figure 3).

```
<xsl:if test="//packagedElement[@b:type='uml:Class']">
                <xsl:for-each select="//packagedElement[@b:type='uml:Class']">
                  <xsl:variable name="nameC" select="@name"/>
                  <xsl:variable name="idC" select="@b:id"/>
                  <UML:Class xmi.id = '-{$idC}'
                    name = '{$nameC}' visibility = 'public' isSpecification = 'false'
isRoot = 'false'
                    isLeaf = 'false' isAbstract = 'false' isActive = 'false'>


                    <UML:Classifier.feature>
                      <xsl:if
test="//packagedElement[@name=$nameC]/ownedAttribute">
                        <xsl:for-each
select="//packagedElement[@name=$nameC]/ownedAttribute">
                          <xsl:variable name="nameAtt" select="@name"/>
                          <xsl:variable name="nameV" select="@visibility"/>
                          <xsl:variable name="idA" select="@b:id"/>
                          <xsl:variable name="idM" select="uuid:randomUUID()"/>
                          <xsl:variable name="idMr" select="uuid:randomUUID()"/>
 <xsl:variable name="typ" select="type/@href"/>
                          <UML:Attribute xmi.id = '-{$idA}'
                            name = '{$nameAtt}' visibility = '{$nameV}' isSpecification
= 'false'
                            ownerScope = 'instance' changeability = 'changeable'
targetScope = 'instance'>
                            <UML:StructuralFeature.multiplicity>
                              <UML:Multiplicity xmi.id = '-{$idM}'>
                                <UML:Multiplicity.range>
                                  <UML:MultiplicityRange xmi.id = '-{$idMr}'
                                    lower = '1' upper = '1'/>
                                </UML:Multiplicity.range>
                              </UML:Multiplicity>
                            </UML:StructuralFeature.multiplicity>

                          </UML:Attribute>
                        </xsl:for-each>
                        </xsl:if>
```

Figure 9: Modelio to ArgoUML: Transformation of the class Member and its attributes.

After defining the correspondent rules for each tag, we have developed the transformation process that allows the exchange between UML tools. The table 6 shows some rules.

Table 6: Table of correspondences of the xslt rules

MOD2ARG	ARG2MOD
<UML:Package xmi.id = '-{$idpack}' name = '{$nompack}' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>	<packagedElement xmi:type="uml:Package" xmi:id="_{$idpack}" name="{$nompack}">
<xsl:for-each select="//packagedElement[@b:type='uml:Class']"> <xsl:variable name="nameC" select="@name"/> <xsl:variable name="idC" select="@b:id"/> <UML:Class xmi.id = '-{$idC}' name = '{$nameC}' visibility = 'public' isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' isActive = 'false'>	<xsl:for-each select="//a:Package/a:Namespace.ownedElement/a:Class"> <xsl:variable name="nameC" select="@name"/> <xsl:variable name="idC" select="@xmi.id"/> <packagedElement xmi:type="uml:Class" xmi:id="_{$idC}" name="{$nameC}">
<xsl:for-each select="//packagedElement[@name=$nameC]/ownedAttribute"> <xsl:variable name="nameAtt" select="@name"/> <xsl:variable name="nameV" select="@visibility"/> <xsl:variable name="idA" select="@b:id"/> <xsl:variable name="idM" select="uuid:randomUUID()"/> <xsl:variable name="idMr" select="uuid:randomUUID()"/> <xsl:variable name="typ" select="type/@href"/> <UML:Attribute xmi.id = '-{$idA}' name = '{$nameAtt}' visibility = '{$nameV}' isSpecification = 'false' ownerScope = 'instance' changeability = 'changeable' targetScope = 'instance'> <UML:StructuralFeature.multiplicit> <UML:Multiplicity xmi.id = '-{$idM}'> <UML:Multiplicity.range> <UML:MultiplicityRange xmi.id = '-{$idMr}' lower = '1' upper = '1'/> </UML:Multiplicity.range> </UML:Multiplicity>	<xsl:for-each select=" //a:Package/a:Namespace.ownedElement/a:Class[@name=$nameC]/a:Classifier.feature/a:Attribute"> <xsl:variable name="nameAtt" select="@name"/> <xsl:variable name="nameV" select="@visibility"/> <xsl:variable name="idA" select="@xmi.id"/> <xsl:variable name="idM" select="uuid:randomUUID()"/> <xsl:variable name="idMr" select="uuid:randomUUID()"/> <ownedAttribute xmi:type="uml:Property" xmi:id="_W{$idA}" name="{$nameAtt}" visibility="{$nameV}" type="_{$idM}" isUnique="false"> </ownedAttribute>

The component XMI FILE results as obtained and allows to present the transformed file, this file is then imported into the target tool and the class diagram is visualized as it appears in the following figure:

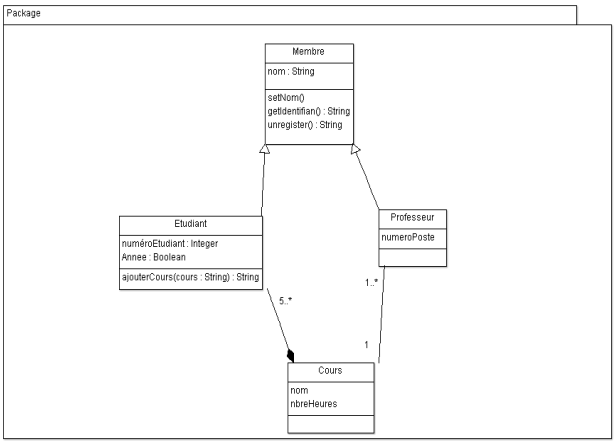


Figure.10: Class Diagram of the target tool

CONCLUSION

The Unified Modeling Language (UML) is a standard language for specifying, viewing, building and documenting software system architectures. UML modeling tools use XMI files for storing and exchanging UML models. The interoperability of UML modeling tools can make UML models reusable and extensible. However, we found that the incompatibility of the tags of the XMI files of these tools was the cause for the problem of interoperability in the UML diagrams.

In this paper, we have studied the interoperability between the UML modeling tools: first by presenting the information about the versions of both UML and XMI of each tool, and then by exchanging the XMI files generated by three tools: Modelio, ArgoUML and Bouml. However, the exchange of models could not be accomplished, which led us to conclude that the writing of XMI tags which varies from one tool to another was the cause of the problem of syntactic interoperability. So, we used the XSLT Template transformation approach in the exchange between Modelio and ArgoUML in order to solve this problem.

Concerning future work, we intend to transform the XMI files using the hybrid transformation through the ATL language to compare this solution with the XSLT template transformation. This comparison will allow us to test and choose the best solution to solve the interoperability problem between UML modeling tools.

REFERENCES

[1] "Model To Text." [Online]. Available: http://www.eclipse.org/modeling/m2t/.

[2] Quyet Thang, "Model Transformation Reuse : A Graph-based Model Typing Approach," Universit é europé enne de Bretagne, Université de Rennes 1, 2012.

[3] N. Cuong and X. Qafmolla, "Model transformation in web engineering and automated model driven development," Int. J. Model. Optim., vol. 1, no. 1, 2011.

[4] M. Dehayni and K. Barbar, "Some Model Transformation Approaches: a Qualitative Critical Review," J. Appl. …, vol. 5, no. 11, pp. 1957–1965, 2009.

[5] Number, OMG Document. OMG Unified Modeling Language TM (OMG UML) Version 2.5. no. March, 2015.

[6] Number, OMG Document. XML Metadata Interchange (XMI) Specification. no. June,2015.

[7] ''ArgoUML User Manual : A tutorial and reference description'' by Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, Jeremy Bennett, Linus Tolke, and Michiel van der Wulp

[8] "BoUML." [Online]. Available: http://www.bouml.fr/.

[9] Sparx systems, "Enterprise architect." [Online]. Available: http://www.sparxsystems.com.au/products/ea.

[10] ''Visual Paradigm'' [Online]. Available : https://www.visual-paradigm.com/

[11] Modeliosoft, the open source modeling environment, "Modelio." [Online]. Available: https://www.modelio.org/.

[12] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois, and F. Terrier, "Papyrus UML: an open source toolset for MDA," in *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (*ECMDA-FA 2009), 2009, pp. 1–4.

[13] Topcased [Online]. Available: https://www.polarsys.org/topcased

[14] comsoft-direct, "PowerAMC." [Online]. Available: http://www.comsoft-direct.fr/poweramc.

[15] '' eclipse free uml'' [Online]. Available : https://marketplace.eclipse.org/category/free-tagging/free-uml-eclipse

[16] N. Magic, Inc., MagicDraw: Architecture Made Simple. 2011.

[17] Zarour, Karim. L'interopérabilité des systèmes d'information médicaux : une approche basée agent. s.l. : Université Mentouri de Constantine.

[18] Exploring Differences in Exchange Formats – Tool Support. Jiang, Juanjuan. s.l. : Institute of Software System,Tampere University of Technology, Finland.

[19] *System of Systems Information Interoperability using a Linked Dataspace.* Curry, Edward. s.l. : Digital Enterprise Research Institute,National University of Ireland, Galway.

[20] *System-of-systems support — A bigraph approach to interoperability and emergent behavior.* Chris Stary, Dominik Wachholder. s.l. : Department of Business Information Systems — Communications Engineering, Johannes Kepler University Linz, Altenberger Strabe 69, 4040 Linz, Austria.

[21] On the interoperability of model-to-model transformation languages. Frederic Jouaulta, Ivan Kurtevc. s.l. : Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham AL 35294-1170, United States, 2007.

[22] An Approach to Identify and Manage Interoperability. Gaurav Bansal, Deepak Vijayvargiya, Siddhant Garg, and Sandeep Kumar Singh.