Supplementary Appendix A

The ModelGenerator is the main package of the XML Flat parser that contains a set of classes responsible for interpreting archetypes and transforming Flat XMLs to ISO 13606 EHR XML Extracts and vice-versa. It performs the transformation process based on the classes of the ISO 13606 RM and Archetype Model and the BaseX component for the XQuery processing. ►**Fig. A1** shows the classes model for the ModelGenerator package that includes the IBIME EN13606 component.[1]

The classes of the ModelGenerator has these purposes:

- XMLFlatModel: Create a template of the Flat XML from the referenced archetype that serves as an example of the possible structures of the Flat XML;
- XMLFlatModelService: RESTful service that allows the generation of the XML Flat model file through the XMLFlatModel class;
- XSDGenerator: Creates an XSD file that allows validation of the constraints and elements hierarchy contained in an archetype; see Appendix B;
- XSDGeneratorService: RESTful service that allows generation of an XSD file through the XSDGenerator class;
- XMLFlatConversor: Generates an XQuery script to translate an EHR Extract to XML flat; see Appendix D;
- XMLFlatConversorService: RESTful service that allows the translation of an EHR Extract to XML flat based on the XQuery script generated through the XMLFlatConversor class;
- ExtractConversor: Generates an XQuery script to translate an XML Flat to EHR Extract; see Appendix C; and
- ExtractConversorService: RESTful service that allows translation of an XML Flat to EHR Extract based on the XQuery script generated through the ExtractConversor class.

The Java code for this package of classes is available in Codes at https://github.com/pocppsus/repository.

Supplementary Appendix B

The XSD file used to validate the XML Flat file is generated through the XSDGeneratorService that triggers the XSDGenerator class. The XSD file is generated based on the same archetypes that were used to generate the XML flat file and reflects the ISO 13606 RM. ►**Fig. B1** shows an excerpt of the Java code of the main section of the XSDGenerator class.

At line #5, it is possible to see the validation of the type of ISO 13606 RM object (COMPOSITION, SECTION, ENTRY OR CLUSTER), and in line #10, this XML tag is generated:

- <xs:element name = "paciente" minOccurs = "1" maxOccurs = "1"/>

If the object type is ELEMENT - see line #20 - the tag considers the ISO 13606 datatype for the expected data, and these tags are generated:

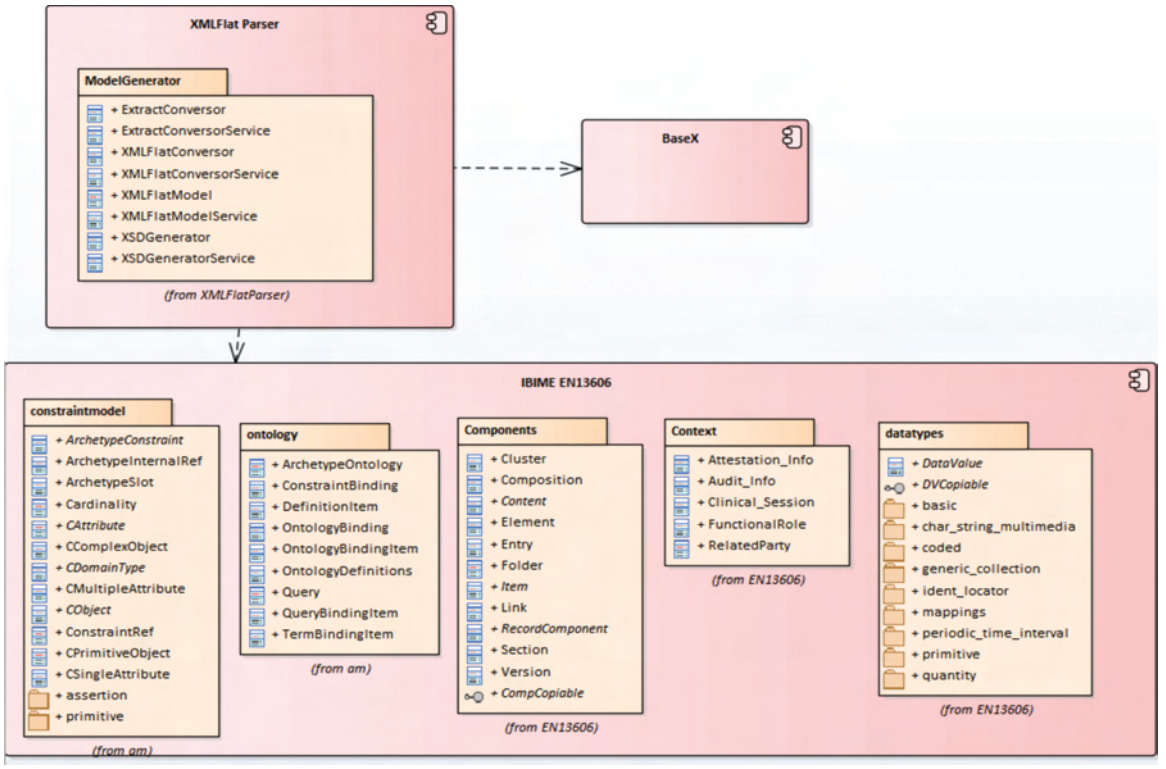

Fig. A1 ModelGenerator classes diagram.

[1] *This component was developed by the Biomedical Informatics Group (IBIME). IBIME is part of the Institute for Applications of Advanced Information and Communication Technologies (ITACA Institute) located at the Technical University of Valencia (UPV), Spain.*

```
protected void printCObject(CObject cObject, int indent, Out out, String existence, String cardinality,
        boolean isOrdered, boolean isUnique) throws IOException {
    if (cObject instanceof CComplexObject) {
        String ocurrences = printCardinalityInterval(cObject.getOccurrences());
        if (cObject.getRmTypeName().compareTo(EN13606Types.COMPOSITION.getName()) == 0
                || cObject.getRmTypeName().compareTo(EN13606Types.ENTRY.getName()) == 0
                || cObject.getRmTypeName().compareTo(EN13606Types.SECTION.getName()) == 0
                || cObject.getRmTypeName().compareTo(EN13606Types.CLUSTER.getName()) == 0) {
            if (!cObject.getArchetype().getNodeText(cObject.getNodeID()).isEmpty()) {
                out.writeln("<xs:element name=\""
                        + printNodeName(cObject.getArchetype().getNodeText(cObject.getNodeID())) + "\" "
                        + ocurrences + " >");
                out.writeln("<xs:complexType>");
            }
            printCComplexObject((CComplexObject) cObject, ++indent, out, "", "");
            if (!cObject.getArchetype().getNodeText(cObject.getNodeID()).isEmpty()) {
                out.writeln("</xs:complexType>");
                out.writeln("</xs:element>");
            }
        } else if (cObject.getRmTypeName().compareTo(EN13606Types.ELEMENT.getName()) == 0) {
            out.writeln("<xs:element name=\""
                    + printNodeName(cObject.getArchetype().getNodeText(cObject.getNodeID()))+ "\" type=\""
                    + getMedicalDataObject((CComplexObject) ((CComplexObject) cObject).getAttribute("value")
                            .getChildren().get(0), indent + 3, out, "$element") + "\" " + ocurrences + "/>");
        }
    }
}
```

Fig. B1 An excerpt of Java code of the main section of the XSDGenerator class.

- <xs:element name = ”carater_da_internacao” type = ” xs:string” minOccurs = ”1” maxOccurs = ”1”/>
- <xs:element name = ”data_e_hora_da_internacao” type = ”xs:dateTime” minOccurs = ”1” maxOccurs = ”1”/>

The XSD file follows the same hierarchy of the archetype, aiming to allow validation of the structure and the order in which the elements of an XML Flat are expected. ►**Fig. B2** shows an XSD file excerpt exemplifying the XSD validation process.

The Java code for this RESTful service and class is available in Codes at https://github.com/pocppsus/repository.

Supplementary Appendix C

The XQuery script that translates an XML Flat to EHR Extract is generated through the ExtractConversorService RESTful service and the ExtractConversor class. The generated XQuery script interprets the XML Flat nodes through the transformXmlFlat method. It generates EHR Extract structures according to these XML Flat nodes. ►**Fig. C1** shows an excerpt of the XQuery script.

►**Fig. C2** shows an example of how an XML Flat node is transformed through the transformXmlFlat method.

Additional functions included in the transformXmlFlat method are:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema elementFormDefault="qualified" id="CEN-EN13606-COMPOSITION.Bres_Sumario_Alta_Obstetrica_Materna.v2.xsd">
<xs:element name="sumario_alta_obstetrica">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="cod_sistema" minOccurs="1" maxOccurs="1"/>
            <xs:element name="paciente" minOccurs="1" maxOccurs="1"/>
            <xs:element name="mulher" minOccurs="1" maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="informacoes_de_auditoria" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                                <xs:all>
                                    <xs:element name="responsabilidade" minOccurs="0" maxOccurs="1">
                                        <!-- Composer -->
                                        <xs:complexType>
                                            <xs:all>
                                                <xs:element name="organizacao_de_saude" minOccurs="0" maxOccurs="1"/>
                                                <!-- Helthcare facility -->
                                                <xs:element name="responsavel" minOccurs="0" maxOccurs="1"/>
                                                <!-- Performer -->
                                            </xs:all>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="nivel_sensibilidade" minOccurs="0" maxOccurs="1"/>
                                    <!-- Sensistivity -->
                                    <xs:element name="synthesised" minOccurs="0" maxOccurs="1"/>
                                    <xs:element name="territory" minOccurs="0" maxOccurs="1"/>
                                    <xs:element name="categoria_informacao" minOccurs="0" maxOccurs="1"/>
                                    <xs:element name="outros_participantes" minOccurs="0" maxOccurs="1">
                                    ...
```

Fig. B2 Excerpt of an generated XSD file.

```
declare function local:transformXmlFlat($transformXmlFlat)
{
    let $codSistema := data($transformXmlFlat/*:raiz/*:cod_sistema)
    let $subjectOfCare := data($transformXmlFlat/*:raiz/*:paciente)
    let $startDate := format-dateTime(current-dateTime(),"[Y0001]-[M01]-[D01]T[H01]:[m01]:[s01]")
    return  <EHR_EXTRACT >
                <ehr_system>
                <extension>{$codSistema}</extension>
                <root><oid>2010.05.24.1.10000</oid></root>
                </ehr_system>
                <ehr_id>
                <extension>{$codSistema}</extension>
                <root><oid>2010.05.24.1.10000</oid></root>
                </ehr_id>
                ...
                <all_compositions>
                ...
```

Fig. C1 Example 1: excerpt of the XQuery script - transformXMLFlat method.

- declare namespace uuid = "java: java.util.UUID": responsible for generating an rc_id unique identifier;
- declare function location: recordComponent($pRecoInfo): responsible for generating audit information for the classes. It uses the following additional methods:

o declare function location: sessionTime ($duration):
- generates the date and time for the clinical encounter or documentation session occurred;

o declare function location: subjectOfCategory($codeValue):
- generates the relationship category of person or object about whom the information in this relates to the subject of care;

◦ declare function location: InfoProvider($performer, $healthcareFacillity):
- generates agent that has provided the information documented;

◦ declare function location: committal($sendInformation):
- generates the agent, date and time, the EHR system and revision into which the data were sent;

◦ declare function location: composer($composer):
- generates the agent responsible for creating, synthesizing or organizing information that is sent to an EHR; and

◦ declare function location: otherParticipations($pOther):
- generates the other agents who have contributed with clinical encounters.

The Java code for this method is available in Codes at https://github.com/pocppsus/repository .

Supplementary Appendix D

The XQuery script that converts an EHR Extract XML to a Flat XML is generated through the XMLFlatConversorService service and XMLFlatConversor class. The XQuery script interprets the structural nodes of the EHR extract XML and iteratively invokes methods generating the XML Flat. ►**Fig. D1** illustrates the main methods of the XMLFlatConversor class.

Each method returns an XML element that may contain other elements according to the method that is used. Fig. D2 shows an example of a root element that its return calls other local method: content($extract/*: all_compositions/*:content).

For example, when the local:content method is called, it returns an element that will have either a member or members items as illustrated in ►**Fig. D3**.

```
{local:recordComponent($transformXmlFlat//*:sumario_alta_obstetrica/*:informacoes_de_auditoria, 'COMPOSITION')}
{for $section in $transformXmlFlat//*:sumario_alta_obstetrica/*:mulher
    return
        <content xsi:type="SECTION">
            <name xsi:type="SIMPLE_TEXT">
                <originalText>Mulher</originalText>
            </name>
            <archetype_id>CEN-EN13606-COMPOSITION.Bres_Sumario_Alta_Obstetrica_Materna.v2/at0001</archetype_id>
            <rc_id>
                <extension>{data(uuid:randomUUID())}</extension>
                <root>
                    <oid>2010.05.24.1.10003</oid>
                </root>
            </rc_id>
            {local:recordComponent($section/*:informacoes_de_auditoria, 'SECTION')}
            {for $entry in $section/*:caracterizacao_da_internacao
                return
                    <members xsi:type="ENTRY">
                        <name xsi:type="SIMPLE_TEXT">
                            <originalText>Caracterização da internação</originalText>
                        </name>
                        <archetype_id>CEN-EN13606-COMPOSITION.Bres_Sumario_Alta_Obstetrica_Materna.v2/at0015</archetype_id>
                        <rc_id>
                            <extension>{data(uuid:randomUUID())}</extension>
```

Fig. C2 Example 2: excerpt of the XQuery script - transformXMLFlat method.

```
declare function location: extract ($extract):
    responsible for interpreting the root document, the clinical extract;
declare spot function: recordComponent($pRecord):
    responsible for interpreting audit information for classes;
declare function location: content($pContent):
    responsible for interpreting the CONTENT which contains an set of
    SECTIONs and ENTRYs that are part of an COMPOSITION;
declare function location: members($pMembers):
    responsible for interpreting a MEMBER which contain other
    SECTIONs and/or ENTRYs that are part of an SECTION;
declare spot function: items($pItems):
    responsible for interpreting an ITEMS which contain other
    CLUSTERs and/or ELEMENTs that are part of an ENTRY;
declare function location: parts($pParts):
    responsible for interpreting an PARTS which contain other
    CLUSTERs and/or ELEMENTs that are part of an CLUSTER;
declare spot function: dataType($pValue):
    responsible for interpreting an ELEMENT and your DATA TYPE which contain
    the recorded value that are part of an ELEMENT.

```

Fig. D1 Example 1: an excerpt of the methods of the of the XMLFlatConversor class for interpreting EHR extract XML.

```
declare function local:extract($extract)
{
    ...
    return <SumarioAltaObstetrica id="{$rcId}">
        <CodSistema>{$codSistema}</CodSistema>
        <Paciente>{$subjectOfCare}</Paciente>
        ...
        {local:content($extract/*:all_compositions/*:content)}
    </SumarioAltaObstetrica>
};

```

Fig. D2 Example 2: an excerpt of the methods of the of the XMLFlatConversor class for interpreting EHR extract XML.

The method local:QNAME, line #7, contains a regex responsible for standardizing the format name of the tag that the XML Flat XML can assume.

The Java code for this method is available in Codes at https://github.com/pocppsus/repository.

Supplementary Appendix E

The XSLT transforms the EHR Extract XML nodes to HTML elements. This process starts from the EHR Extract XML root that returns the document header and calls the

```
declare function local:content($pContent)
{
    for $content in $pContent
        let $name := data($content/*:name/*:originalText)
        let $members := $content/*:members
        let $items := $content/*:items
        return let $el := element { local:Qname($name)}
            {
                if(empty($members))
                    then local:items($items)
                    else if(empty($items))
                    then local:members($members)
                    else ""
            }
            return copy $je := $el
            modify insert node local:recordComponent($content) as first into $je
            return $je
};
```

Fig. D3 Example 3: an excerpt of the methods of the of the XMLFlatConversor class for interpreting the CONTENT class.

elementIterator method. It is performed through the initial method described in the ►**Fig. E1**.

The elementIterator method is responsible for going through the child nodes of a node and trigger the chooseElementType method to choose which template to apply, making iteratively the layout that a structure and its elements should assume. See ►**Fig. E2**.

The definition of which template will be used is made by the chooseElementType method that verifies the type of node being interpreted and choose which template should be applied. See ►**Fig. E3**.

For each selected element in chooseElementType a template is triggered to return a defined HTML layout and calling again the elementIterator method for child nodes if any. There are templates for section, entry, cluster, element and datatype structures.

The Java code for this method is available in Codes at https://github.com/pocppsus/repository.

```
1  <xsl:template match="/">
2      <xsl:apply-templates select="EHR_EXTRACT"/>
3  </xsl:template>
4  <!-- produce browser rendered, human readable clinical document -->
5  <xsl:template match="EHR_EXTRACT">
6
```

Fig. E1 An excerpt of the XSTL root for interpreting EHR extract XML.

```
1  <xsl:template name="elementIterator">
2      <xsl:param name="iterator"/>
3      <xsl:for-each select="$iterator">
4          <xsl:call-template name="chooseElementType">
5              <xsl:with-param name="chooseElementTitle"cselect="name/originalText"/>
6          </xsl:call-template>
7      </xsl:for-each>
8  </xsl:template>
9
```

Fig. E2 An excerpt of the XSTL element iterator for interpreting elements.

```
1  <xsl:template name="elementIterator">
2      <xsl:param name="iterator"/>
3      <xsl:for-each select="$iterator">
4          <xsl:call-template name="chooseElementType">
5              <xsl:with-param name="chooseElementTitle"cselect="name/originalText"/>
6          </xsl:call-template>
7      </xsl:for-each>
8  </xsl:template>
9
```

Fig. E3 An excerpt of the XSTL choose element type.