Automatic generation of Web Users Interfaces using a Model-Driven Approach.

Thierry NOULAMO, Bernard FOTSING TALLA, Jean-Pierre LIENOU

Abstract— One of the main objectives of software engineering is to evaluate and improve the quality and productivity of software. Productivity is therfore a determining factor to take into account during software development.This paper focuses on the automatic production, using the Model-Driven Engineering (MDE) approach, of software systems in general and human-machine interfaces of interactive applications in particular. Our method aims to produce two design Meta-Models: a source Meta-Model called “DD_IHM” ("Description Diagram for Human-Machine Interfaces"), for the production of HMIs by exploiting the graphic richness proposed by UML, and a target Meta-Model called “CGFP” (Context Grammar for PEAR), specific to the PHP language described by the productions of the underlying context-free grammar. We propose then a set of generic rules written in the QVT language, for the transformation of a model conforming to the source Meta-Model into a model conforming to the target Meta-Model. We validate our approach on a simple example of a currency converter.

Index Terms— Web Applications, HMI, Models Transformation, Context-free Grammar, QVT, Software productivity.

---

1 INTRODUCTION

Productivity is a complex concept that should be measured or defined in order to improve the software quality [1]. The productivity in ISO 9126 is a quality factor of the software products and is defined as “the capability of the software product to enable users to expend appropriate amount of resources in relation to the effectiveness achieved in a specified context of use” [2]. In recent years it becomes clear that if we want to produce quality software, it is necessary to focus on both the products of software and the process used for software development [3]. We propose in this paper an approach to improve the software productivity.

Nowadays, user interfaces are complex software components that play a vital role in the implementation of applications. Their development requires the use of a process that integrates the visual models and the standardized notations for this visualization. The use of Multi-Tier architecture, for the design of applications favors the production of reusable and easily maintainable systems [4]. The development of web applications in particular can be done according to a Three-tier architecture : the presentation tier still called the external layer, which corresponds to the construction of human-machine interactions; the application tier, still called functional layer, which corresponds to the construction of the semantic component, and the data tier called the inner layer. We are interested in this work by the presentation layer.

The complexity of user interfaces of modern systems requires, like all other phases of the systems development process, a methodology and a good modeling technique. The two main approaches to achieving these types of systems are the task-based and the model-based approach. We are interested in this work by the second approach. According to Schlungbaum [5], the knowledge involved in the implementation of interactive systems can be represented by models. Our attention is focused on the production of an operationnal human-machine interaction models.

Many commercial tools, often referred generically as User Interface Management Systems (UIMS), are currently being proposed to make easier the implementation of the interactive systems presentation layer by the non-specialist programmers [6].

However, the use of such tools does not promote the automation of the entire process of developing interactive systems and are for the most part proprietary.

Model Driven Engineering (MDE) is today at the heart of the system development activities in the industry [7], [8], [9] and promotes the model of the systems as the primary artefact of the developmental principle [7]. However, the models used must be operational so that an implementation can be generated systematically.

The objective of this work is to produce a human machine interface (HMI) Meta-Model, which is intended to be operational, describing precisely the interactions expected in the interactive system. This paper adds graphical interface design elements to the design elements proposed in [10][11]. In [11], authors proposed an approach base on two level transformation. The interaction diagram obtained is translated in this work into a PHP-specific Meta-Model PHP by using a single model transformation generics rules, implemented in the QVT language.

The work is structured as follows. In section 2, we review the works that has been done on HMI modeling in the interactive systems development process. In section 3, we present our design approach. In Section 4, we describe the source Meta-Model, the target Meta-Models and we propose a generic description of the transformation rules from the source model to the target model that we implement with QVT. Section 5 shows through an example the automatic generation of the interface of a simple currency converter. In section 6, we presents the physical architecture of our approach and highlights the components necessary for the implementation of an HMI according to this approach. We conclude in section 7 by identifying some future work.

2 EXPERIENCE IN IMPLEMENTING HMIS.

The need to realize the interfaces of an application with the same rigor as that granted to the application itself becomes essential in critical systems in particular and in computer applications in general for reasons of cost, reuse, maintenance, reliability and usability. Several notations make possible the expression of user needs concerning the human-machine interaction: AMD (Analytical Method of Description) [12], HTA (Hierarchical Task Analysis) [13], UAN [14] and its extension XUAN [15] and the notation ConcurTaskTrees (CTT1 [16]). All these notations constitute the starting base of realization of an HMI. However, they are mostly user centric and are far from the operational computer models. Whithout any doubt, the model-based development systems will emerge. The architecture models derived from these systems assume that an interactive system has an interface portion and a functional core portion that refers to the domain of the application. Several models of architecture exist in the literature: layered models such as Seeheim [17] and Arch [18], agent models such as MVC (Model, View, Controller), PAC (Presentation, Abstraction Control) [19], [20], [21], [4] and mixed models that attempt to take advantage of the respective advantages of the two previous categories.

We are interested in this paper in agent models. They all have as a common element the interface between the functional kernel and the user, called "view" in the MVC model and "Presentation" in the PAC model. However, no specification is made of the details of this layer. Our goal is to automate the development of the user interfaces. This overall objective is broken down into several specific objectives: to propose, in the form of a Meta-Model, a set of component patterns intended for the specification of the presentation layer; to propose a HTML QuickForm profile for the PHP language called "CGFP" and, finally, to propose a set of rules for the transformation of an instance of the current source Meta-Model to an instance of the target Meta-Model.

3 DESIGN APPROACH

The figure 1 presents the different design steps of our approach.

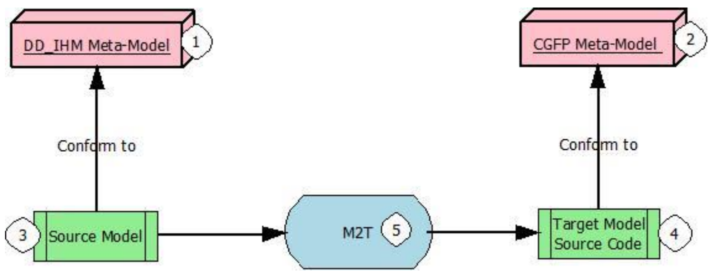


Fig. 1. The different stages of the design approach

We use the "DD-IHM" meta-model (1) to build a source model (3). The Model to Text transformation (5) is implemented to transform a source model into a target model (4) that is conforming to "CGFP" Meta-Model (2).

4 DESCRIPTION OF META-MODELS AND THE TRANSFORMATION RULES

The Unified Modeling Language (UML) [22], [23] has established itself as a de facto standard for the modeling of systems by following an object-oriented approach.

We use it, in this section, to represent the different metaclasses constituting the elements of our source Meta-Model. We also present in the form of a concrete syntax of the context-free grammar the description of the elements of the target Meta-Model and give the generic specifcation of the associated transformation rules in the QVT language.

4.1 The source Meta-Model: DD-IHM

The source Meta-Model that we propose covers the significant concepts for modeling user interfaces. We use the class diagram, the activity diagram and the transitional state diagram of the UML Meta-Model [18] to build new design elements. This allows us to present the elements of the HMI patterns and the DTD describing the storage format.

The Figure 2 models in a UML class diagram the main elements that can be found in an HMI. In particular, the element "FormClass" that represents a form with its attributes is composed mainly of two sub-classes: the class "Node" and the class "Attribut". Node is composed of three classes: "Data", "Rules" and "groupeELT". The class "Data" is similar to label element in HTML language, it allows to paste a label to an element of the form. The "Rules" class materializes the set of validation constraints that can be assimilated to a component of the form. A node is composed of at most one element group. In a group we can have several subgroups. The element "Menu", as its name suggests, allows to build the components that are used to navigate in the application. The menu can be vertical or horizontal. An HMI can contain several menus. Recall that our main concern is the production of models from which we can automatically generate a PHP implementation of interactive systems HMIs. For this purpose, we use a subset of the UML modeling language, to which we have added a set of concepts to adapt the method to a composition approach of elements of the Meta-Model. The class diagram is the basis of this proposition. Each node of the Meta-Model is associated to the communication ports (input and output) to obtain a dynamic object that is an instance of a class of the basic diagram (figure 2). The composition of the "Dynamics Objects" using the activities diagram nodes include in the Meta-Model give us a new diagram called "Dynamic Objects Diagram" (DoD). This allows the modeling of an HMI by connecting inputs and outputs of appropriately selected dynamic objects. An HMI is considered as a large object, using the above elements as the states. The entire model is stored in the XML format which is conforming to the DTD, given by the listing 1 below, to support its use in a Models Driven Engineering approach.

International Journal of Scientific & Engineering Research, Volume 11, Issue 9, September-2020
ISSN 2229-5518

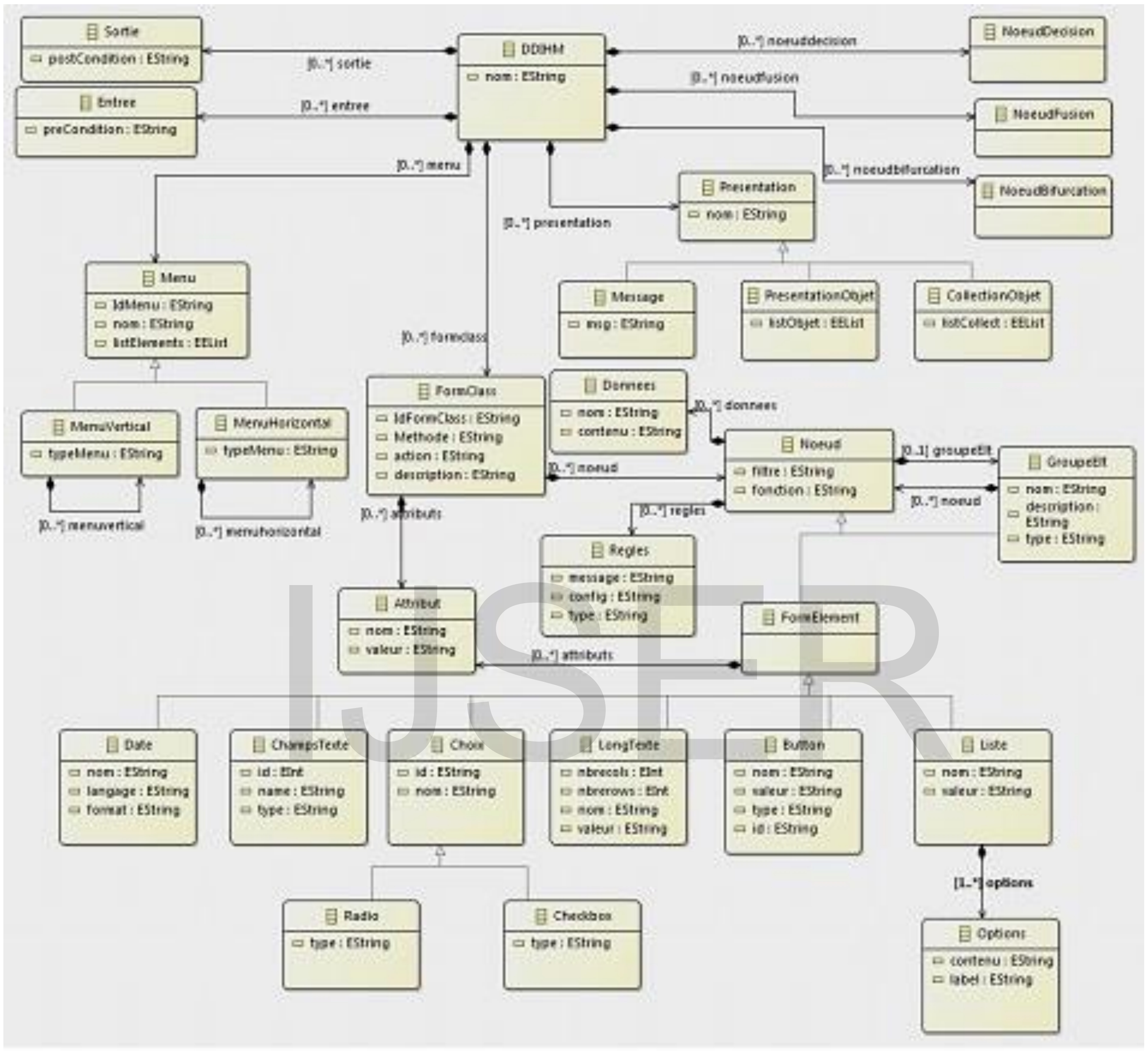
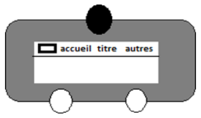
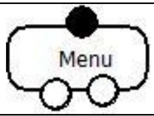
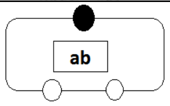
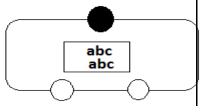
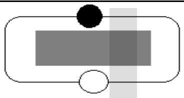
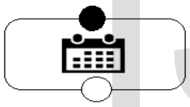
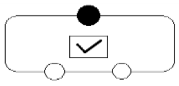
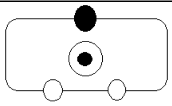
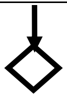
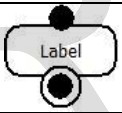


Fig2. DD-IHM Meta-Model class diagram

The table 1 describes the elements of the source Meta-Model as well as their representation in the form of DoD.

TABLE 1
SEMANTICS OF HMI PATTERNS DESCRIBING THE SOURCE META-MODEL

Element's names	Representation	Semantics
Form_Class		Main element that encapsulates other elements
Group_Elt		Group element
Menu Node		
Field_Text		Allows to create a HTML Text Field
Long_Field_Text		Allows to create a HTML Long Text Field
Bouton		Allows to create a HTML Button
Date		Allows to create a HTML Date
CheckBox		Allows to create a HTML CheckBox
Radio		Allows to create a HTML Radio Field
Initial_Node		Beginning of the diagram, UML Element
End_Node		End of the diagram, UML Element
Transition_Node		Materializes the transition between two for elements, UML Element
Test_Node		Use as a bridge between two or more form elements, UML Element
Synchronization_ Node		Materialize the synchronazation between the form elements, UML Element
Label Node		Use to create the label in a forme
Condition Node	**[Cond]**	Logical condition associated to the decision nodes

4.2. The target Meta-Model : CGFP

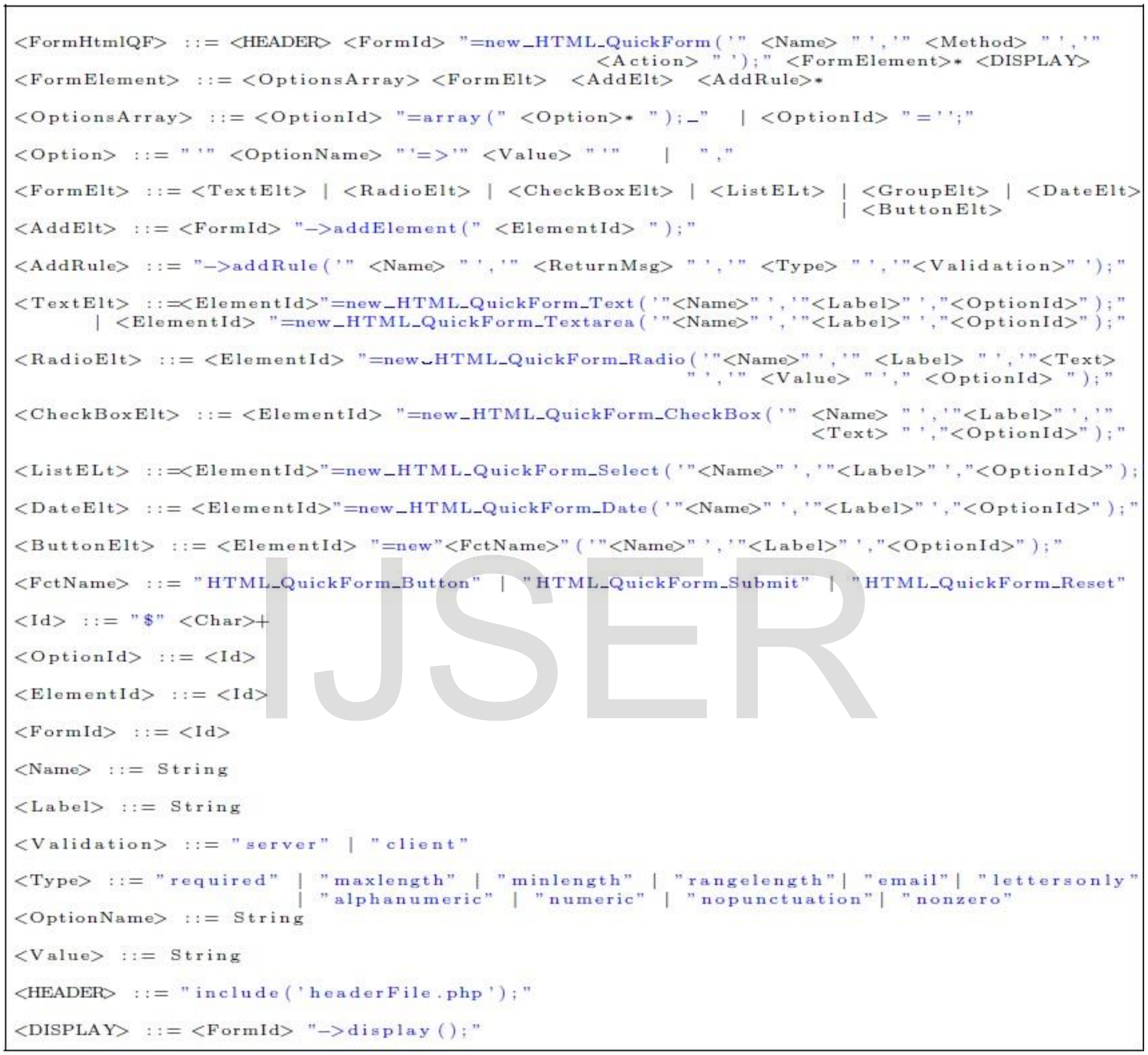
The target Meta-Model is an abstract description of the HTML QuickForm code for generating an HMI in PHP (which is a typical HTML form). This description is given in the form of productions (in Backus Naur Form - BNF notation) of the underlying context-free grammar (see Listing 2 below). Recall that a context-free grammar consists of a tuple G = (V, T, P, S) where: "V" is the (finite) set of variables (or nonterminals or syntactic categories such as <RadioElt>, <TextElt>, ...). Each variable represents a language, i.e., a set of strings; "T" is a finite set of terminals, i.e., the symbols that form the strings of the language being defined; "P" is a set of production rules that represent the recursive definition of the language and "S" is the start symbol that represents the language being defined. Other variables represent auxiliary classes of strings that are used to help define the language of the start symbol (<FormHTMLQF>).

International Journal of Scientific & Engineering Research, Volume 11, Issue 9, September-2020
ISSN 2229-5518

LISTING 1. EXTRACT OF THE DTD OF THE HMI PATTERNS DESCRIBING THE SOURCE META-MODEL.

```
<?xml version="1.0"? encoding="ISO-8859-1">
<!DOCTYPE DoD_IHM_DIAGRAM[
<!ELEMENT FormNode(BeginNode, labelNode (Node)+, EndNode>
<!ATTLIST DoD_Form Name CDATA #REQUIRED Method CDATA #REQUIRED Action CDATA #REQUIRED>
<!ELEMENT BeginNode EMPTY>
<!ATTLIST BeginNode ID_Elements ID Next_node IDREF #REQUIRED>
<!ELEMENT LabelNode EMPTY>
<!ATTLIST LabelNode ID_Elements ID ID_Next IDREF #REQUIRED>
<!ELEMENT Node (StructNode| FormElem>
<!ELEMENT EndNode EMPTY>
<!ATTLIST EndNode ID_Elements ID  #REQUIRED>
<!ELEMENT StructNode(BeginNode | EndNode | ForkNode | JunctionNode | TestNode
                                                    | TransitionNode | CondNode)>
<!ELEMENT FormElemNode(LabelNode | Menu | Text | RadioNode | CheckBox | List
                                                          | Button | Group)>
<!ELEMENT ForkNode(FormElemNode)+>
<!ATTLIST ForkNode ID_Elements ID Next_node IDREFS #REQUIRED>
<!ELEMENT JunctionNode EMPTY>
<!ATTLIST JuntionNode ID_Elements ID Next_node IDREFS #REQUIRED>
<!ELEMENT TestNode(CondNode, CondNode+)>
<!ATTLIST TestNode ID_Elements ID Next_node IDREFS #REQUIRED>
<!ELEMENT CondNode(FormElem+)>
<!ATTLIST CondNode ID_Elements ID Next_node IDREF Condition DATA #REQUIRED>
<!ELEMENT TransitionNode EMPTY>
<!ATTLIST TransitionNode ID_Elements ID source_node IDREF target_node IDREF #REQUIRED>
<!ELEMENT MenusNode (BeginNode,(MenuNode+,(StructNode),MenuNode+),End_Node)*)>
<!ATTLIST MenuNode ID_Elements ID Name CDATA Value CDATA parent IDREF #REQUIRED>
<!ELEMENT InputElem (Rules)>
<!ATTLIST InputElem ID_Elements ID  #REQUIRED
        Name CDATA #REQUIRED
        Value CDATA
        Type CDATA #REQUIRED
        Multilign CDATA
        NumberOfLigne CDATA
        NumberOfCol CDATA
        ID_Warning IDREF #REQUIRED
        ID_Sucess IDREF #REQUIRED>
<!ELEMENT ButtonElem (Rules)>
<!ATTLIST ButtonElem ID_Elements ID  #REQUIRED
        Name CDATA #REQUIRED
        Value CDATA #REQUIRED
        Type CDATA #REQUIRED
        Action CDATA #REQUIRED
        ID_Warning IDREF #REQUIRED
        ID_Sucess IDREF #REQUIRED>
<!ELEMENT RadioElem (Rules)>
<!ATTLIST RadioElem ID_Elements ID  #REQUIRED
        Name CDATA #REQUIRED
        Value CDATA #REQUIRED
        ID_Warning IDREF #REQUIRED
        ID_Sucess IDREF #REQUIRED>
<!ELEMENT ChecKBoxElem (Rules)>
<!ATTLIST  ChecKBoxElem ID_Elements ID  #REQUIRED
        Name CDATA #REQUIRED
        Value CDATA #REQUIRED
        ID_Warning IDREF #REQUIRED
        ID_Sucess IDREF #REQUIRED>
<!ELEMENT SelectElem (SelectOption*, Rules*)>
<!ATTLIST  SelectElem ID_Elements ID  #REQUIRED
        Name CDATA #REQUIRED
        ID_Warning IDREF #REQUIRED
        ID_Sucess IDREF #REQUIRED>
<!ELEMENT  SelectOption EMPTY>
<!ATTLIST SelectOption Name CDATA #REQUIRED
        Value CDATA
        ID_Next IDREF #REQUIRED>
<!ELEMENT Rules EMPTY>
<!ATTLIST Rules Name CDATA #REQUIRED
        Value CDATA
        ID_Next IDREF #REQUIRED>
]>
```

LISTING 2. PRODUCTIONS OF THE CGFP DESCRIBING THE TARGET META-MODEL.



```
<FormHtmlQF> ::= <HEADER> <FormId> "=new_HTML_QuickForm('" <Name> "','" <Method> "','"
                                        <Action> "');" <FormElement>* <DISPLAY>
<FormElement> ::= <OptionsArray> <FormElt>  <AddElt>  <AddRule>*

<OptionsArray> ::= <OptionId> "=array(" <Option>* ");_"   | <OptionId> "='';"

<Option> ::= "'" <OptionName> "'=>'" <Value> "'"    |   ","

<FormElt> ::= <TextElt> | <RadioElt> | <CheckBoxElt> | <ListELt> | <GroupElt> | <DateElt>
                                                                 | <ButtonElt>
<AddElt> ::= <FormId> "->addElement(" <ElementId> ");"

<AddRule> ::= "->addRule('" <Name> "','" <ReturnMsg> "','" <Type> "','"<Validation>"');"

<TextElt> ::=<ElementId>"=new_HTML_QuickForm_Text('"<Name>"','"<Label>"',"<OptionId>");"
      | <ElementId> "=new_HTML_QuickForm_Textarea('"<Name>"','"<Label>"',"<OptionId>");"

<RadioElt> ::= <ElementId> "=new_HTML_QuickForm_Radio('"<Name>"','" <Label> "','"<Text>
                                               "','" <Value> "'," <OptionId> ");"

<CheckBoxElt> ::= <ElementId> "=new_HTML_QuickForm_CheckBox('" <Name> "','"<Label>"','"
                                                       <Text> "',"<OptionId>");"

<ListELt> ::=<ElementId>"=new_HTML_QuickForm_Select('"<Name>"','"<Label>"',"<OptionId>");

<DateElt> ::= <ElementId>"=new_HTML_QuickForm_Date('"<Name>"','"<Label>"',"<OptionId>");"

<ButtonElt> ::= <ElementId> "=new"<FctName>"('"<Name>"','"<Label>"',"<OptionId>");"

<FctName> ::= "HTML_QuickForm_Button" | "HTML_QuickForm_Submit" | "HTML_QuickForm_Reset"

<Id> ::= "$" <Char>+

<OptionId> ::= <Id>

<ElementId> ::= <Id>

<FormId> ::= <Id>

<Name> ::= String

<Label> ::= String

<Validation> ::= "server" | "client"

<Type> ::= "required" | "maxlength" | "minlength" | "rangelength"| "email"| "lettersonly"
               | "alphanumeric" | "numeric" | "nopunctuation"| "nonzero"
<OptionName> ::= String

<Value> ::= String

<HEADER> ::= "include('headerFile.php');"

<DISPLAY> ::= <FormId> "->display();"
```

4.3. The transformation rules

QVT (Query/View/Transformation) is a standard of languages for model transformation defined by the Object Management Group [24]. The QVT standard defines three model transformation languages. All of them operate on models which are conform to Meta-Object Facility (MOF) 2.0 metamodels. We are interested here by the QVT-Operational model, which is an imperative language designed for writing unidirectional transformations. A model transformation takes as input a model conforming to a source Meta-Model and produces as output another model conforming to a target metamodel.

We first of all declare the source and target Meta-Models types respectively "DD_IHM" and "CGFP", which content the implementation of all the HMI elements in the appropriate formats, as follow:

modeltype DD_IHM uses "http://localhost/hmi2";
modeltype CGFP uses "http://localhost/htmlQF.hs";

The structure of the transformation rules is given as follow:

International Journal of Scientific & Engineering Research, Volume 11, Issue 9, September-2020
ISSN 2229-5518

```
    transformation hmi2htmlQF(in hmi :DD_IHM, out htmlQF :CGFP){
    main(){
    hmi.objects()[hmi::FormNode]->map frmNode2frmHtmlQF();
    hmi.objects()[hmi::TextNode]->map txtNode2txtHtmlQF();
    hmi.objects()[hmi::RadioNode]->map radioNode2radioHtmlQF();
    hmi.objects()[hmi::CheckBoxNode]->map cbNode2cbHtmlQF();
    hmi.objects()[hmi::ListNode] ->map lstNode2lstHtmlQF();
    hmi.objects()[hmi::DateNode]->map dateNode2dateHtmlQF();
hmi.objects()[hmi::ButtonNode] ->map btnNode2btnHtmlQF();
}
...
```

It transforms any input model hmi of type SimpleHMI_MM into the corresponding output model htmlQF of type HtmlQF_MM. The function main() is the entry point of the transformation. The following statement (which is a typical transformation rule)

hmi.objects()[hmi::FormNode] ->map frmNode2frmHtmlQF();

applies frmNode2frmHtmlQF() to each FormNode object in the input model and translates it into the corresponding HTML_QuickForm code.

Every mapping is an operation associating an element from the input model with another element from the output model. For example, to transform a FormNode to FormHtmlQF, the corresponding mapping operation is:

```
mapping hmi::FormNode::frmNode2frmHtmlQF() : htmlQF::FormHtmlQF {
FormHtmlQF
:="include('headerFile.php');"+self.Name+" id = new HTML QuickForm ('"+ self.Name+"','"+ self.Method + "','" + self.Action+ "'); " + hmi::FormNode::frmElements();
}
...
}
```

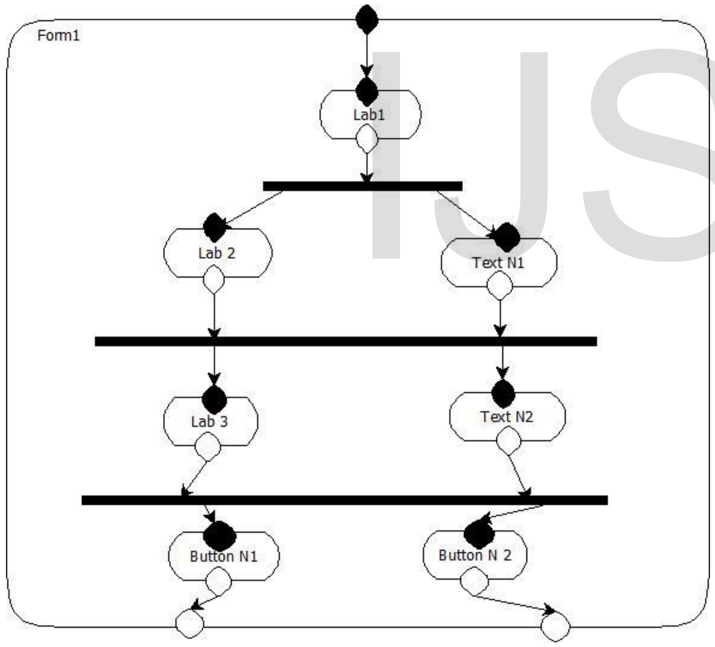


Fig. 3. Model of our case study using DD_IHM

5 Application to a Simple Currency Converter

We present in this section an example of a HMI of a simple currency convertor. The interface is described in XML format with the appropriate elements of the source meta-model as shown in the listing 3.

Listing 3. Source model's XML description of the converter.

```
<FormNode name="frmConv" method="post" action="">
      <BeginNode />
      <LabelNode  id="head" value="Simple␣Converter"/>
      <ForkNode>
            <LabelNode  id="lbl1"  value="XAF:"/>
            <TextNode  id="txtXAF"  value=""/>
      </ForkNode>
      <ForkNode>
            <LabelNode  id="lbl2"  value="USD:"/>
            <TextNode  id="txtUSD"  value=""/>
      </ForkNode>
      <ForkNode>
            <ButtonNode name="btnXAF2USD"
                   value="XAF␣to␣USD" type="submit" />
            <ButtonNode name="btnUSD2XAF"
                   value="USD␣to␣XAF" type="submit" />
      </ForkNode>
      <EndNode />
</FormNode>
```

The application of the transformation rules defined previously generates the HTML QuickForm code as shown by the listing 4 which leads to the user interface for the currency converter of the Figure 4.

International Journal of Scientific & Engineering Research, Volume 11, Issue 9, September-2020
ISSN 2229-5518

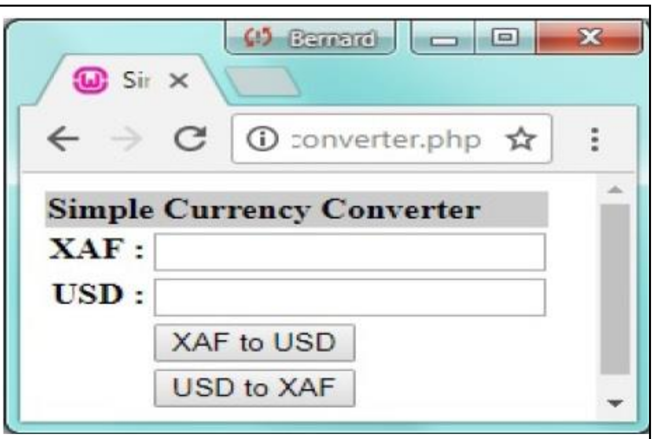


Fig. 4. HMI of the Converter.

LISTING 4. GENERATED HTML QUICKFORM OF THE CONVERTER

```
<?php
    set\_include_path(get_include_path() . ";c:\php\pear");
    require_once "HTML/QuickForm.php";
    require_once "HTML/QuickForm/text.php";
    require_once "HTML/QuickForm/header.php";
    require_once "HTML/QuickForm/submit.php";
    $frmConv_id = new HTML_QuickForm('frmConv', 'post');
    $head_id = new HTML_QuickForm_Header ('head',
                                'Simple␣Currency␣Converter');
    $frmConv_id ->addElement($head_id);
    $txtXAF_id = new HTML_QuickForm_text ('txtXAF', 'XAF:', '');
    $frmConv_id->addElement($txtXAF_id);
    $txtUSD_id = new HTML_QuickForm_text ('txtUSD', 'USD:', '');
    $frmConv_id->addElement($txtUSD_id);
    $btnXAF2USD_id = new HTML_QuickForm_Submit('btnXAF2USD',
                                              'XAFtoUSD', '');
    $frmConv_id->addElement($btnXAF2USD_id);
    $btnUSD2XAF_id = new HTML_QuickForm_Submit('btnUSD2XAF',
                                              'USDtoXAF', '');
    $frmConv_id->addElement($btnUSD2XAF_id);
    $frmConv_id->display();
?>
```

6 PHYSICAL MODEL

Once Pear and the package HTML QuickForm2 installed, we will have a tree structure represented by the figure 5.

 PHP Server: The folder in which the web server and php are installed,

 PEARDIR: The PEAR installation folder,

 HTML: The installation folder for HTML packages,

 QuickForm2: The installation folder of HTML QuickForm2.

The php source file obtained after transforming the "DD_IHM" model is stored in the PEARDIR folder. And can therefore be run via the url "http://localhost/peardir/currency_converter.php". A file named "head.php" containing the definition of the personalization data of the page is placed in the HTML folder of our tree. This file contains the reference of the stylesheet files (quickform.css) and JavaScript (js/quickform.js) both placed in a folder named data, located in the "PEARDIR" folder.

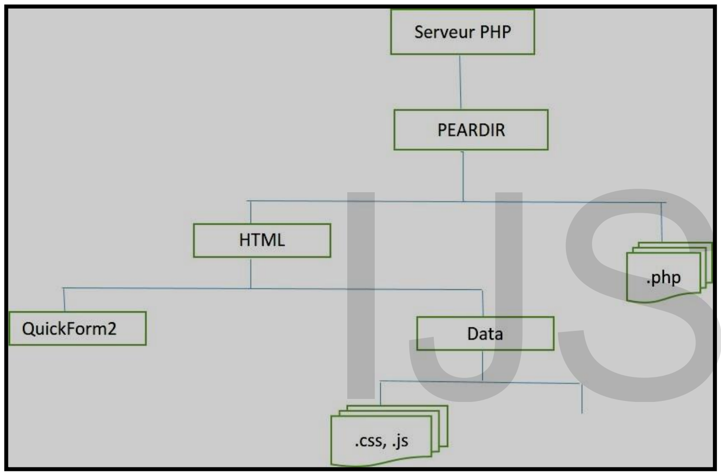


Fig. 5. Deployment architecture

7 CONCLUSION AND FURTHER WORKS

This work focused on the automatic production of the human-machine dialogue layer. It's part of a broader research perspective aimed at fully automating the software production process. Indeed it shows that it is possible to produce 100% implementation of a software system using the IDM approach. We have proposed a set of design patterns that model the HMI components and are used to represent the details of the presentation layer of the Web applications and a target Meta-Model that is an abstract description in the form of HTML_QuickForm code for the HMI generation in PHP. The rules for transforming a model of the source Meta-Model into a target Meta-Model model are described in the QVT language. The results allow us to appreciate the gain in time and efficiency in the production of robust HMI in php. In our future work, we will propose a generalize approach including the data layer and the business layer. This require the extention of the Meta-Models so that they can take into account all the elements of business layer and data layer.

ACKNOWLEDGMENTS

Dr NOULAMO Thierry, Dr FOTSING TALLA Bernard and Dr LIENOU Jean Pierre are Lecturers at FOTSO Victor University Institute of Technology of the University of Dschang.

The authors are involved in the project "WESIP" (Web System Interactive Plat-form), developed within the STIC team of the LAIA laboratory of the University of Dschang. We propose to setting up a platform for the development of interactive Web applications according to a model-based approach.

REFERENCES

[1] Ernandez-Lopez, A. ; Dept. Inf., Univ. Carlos III de Madrid, Legan s, Spain ; Colomo-Palacios, R. ; Garcia-Crespo, A. Productivity in software engineering: A study of its meanings for practitioners:Understanding the concept under their standpoint Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on 20-23 June 2012.

[2] ISO, ISO/IEC TR 9126-4: Information Technology - Product Quality - Part 4: Quality in Use Metrics, Geneva, Switzerland: International Organization for Standardization, 2004.

[3] Anupriya et al., Survey on Various Productivity Measures of Software Development Teams, International Journal of Advanced Research in Computer Science and Software Engineering 4(6), June - 2014, pp. 462-464

[4] G.E. PFA . " User Interface Managment Systems ", Eurographics seminars, Springer Verlag - Berlin, 1985.

[5] SCHLUNGBAUM E., ELWERT T., Automatic user Generation from Declarative Models, In [CADUI 96].

[6] MYERS BRAD A., User Interface Software Tools ", ACM Transactions on Computer Human Interaction. vol. 1, no 2, pp. 64-103, 1995.

[7] R. B. HAILPERN , P. TARR, Model-driven development : The good, the bad and the ugly, IBM Systems Journal, vol. 3,no 45, PP. 1-25, 2006.

[8] D. SCHMIDT, "Guest Editor's Introduction : Model-Driven Engineering ", Computer, vol. 2,no 9, PP. 25-31, Feb. 2006, ISSN 0018-9162.

[9] B. SELIC, Unified Modeling Language version 2.0 - In support of model-driven development, IBM Rational Developer Works,no PP. 1-33 nov. 2005.

[10] E. Andr, C. Choppy, and T. Noulamo. Modelling timed concurrent systems using activity diagram patterns. KSE'14, Springer Advances in Intelligent Systems and Computing, octobre 2014. http://lipn.univ-paris13.fr/ andre/publications fr.php.

[11] T. NOULAMO, B. FOTSING TALLA, M. WANE, L. H. NZOTHIAM TAKOU, A Model-Driven Approach for Developing WEB Users Interfaces of Interactive Systems, International Journal of Computer Trends and Technology (IJCTT) – Volume 68 Issue 4 – April 2020, ISSN: 2231-2803 http://www.ijcttjournal.org Page 33-43

[12] D. L. SCAPIN , C. PIERRET-GOLBREICH, " Towards a Method for Task Description : MAD ", In Work with display units. Elsevier Science Publishers, no North-Holland, 1990.

[13] A.J. DIX, J. FINLAY, G. ABOWD , R. BEALE. Human-Computer Interaction, Prentice Hall, no 1993.

[14] D. RIX , H.R. HARTSON. Developing User Interfaces : Ensuring Usability Through Product Process, Wiley professional computing John Wiley Sons, no inc. NY, USA, 1993.

[15] P. GRAY, D. ENGLAND, , S. MCGOWAN, XUAN : Enhancing the UAN to Capture Temporal Relation Among Actions, Technical report, Department of Computing Science, University of Glasgow, no 2 1994.

[16] FABIO PATERN. Model-Based Design and Evaluation of Interactive Applications , Springer, no 2001.

[17] PFAFF, G. E. (1985). User Interface Management Systems, Workshop on User Interface Management Systems (Eurographic Seminars), Seeheim, Springer-Verlag, no November 1-3, 1985.

[18] UIMS, The UIMS Workshop Tool Developers : A Metamodel for the Runtime Architecture of an Interactive System, SIGCHI Bulletin, no 24, PP. 32-37, 1992.

[19] COUTAZ, J. (1990). " Interfaces Homme-Ordinateur, Conception et Ralisation ", Dunod Informatique, no Paris, 1990

[20] COUTAZ, J., NIGAY, L. (2001). " Architecture logicielle conceptuelle des systmes interactifs (chapitre 7) ", In Kolski, C. (Ed.), Analyse et conception de l'I.H.M., Interaction HommeMachine pour les S.I., Herms Science, vol. 1, no Paris, PP. 207-246, 2001.

[21] OMG, " Unified Modeling Language : Infrastructure - Version 2.5 formal/2015-03-01", OMG, no nov. 2015a, URL http ://www.uml.org/.

[22] Unified Modeling Language : Infrastructure - Version 2.1.2 formal/07-11-04, OMG, no nov. 2007a, URL http ://www.uml.org/.

[23] OMG, Unified Modeling Language : Superstructure - Version 2.1.2 formal/07-11-02, OMGOMG, no . 2007b. URL http ://www.uml.org/.

[24] Bast, Wim; Murphree, Michael; Lawley, Michael; Duddy, Keith; Belaunde, Mariano; Gri_n, Catherine; Sendall, Shane; Vojtisek, Didier; Steel, Jim; Helsen, Simon; Tratt, Laurence; Reddy, Sreedhar; Venkatesh, R.; Blanc, Xavier; Dvorak, Radek; Willink, Ed (January 2011). "Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT)" (pdf). Object Management Group. Retrieved 9 May 2011.