

Original Article

A Model-Driven Approach for Developing WEB Users Interfaces of Interactive Systems

Thierry Noulamo¹, Bernard Fotsing Talla², Merlin WANE³, Lois Hurel Nzothiam Takou⁴

^{1,2,3,4}Department of Computer Engineering, University of Dschang, LAIA, PO Box 134 Bandjoun, CAMEROON

Received Date: 28 February 2020

Revised Date: 15 April 2020

Accepted Date: 16 April 2020

Abstract - Nowadays, User Interfaces are complex software components that play a vital role in the development of interactive applications. Its development requires, as for another phase, the use of a process that integrates the development of visual models and a standardized notation for this visualization. We propose two metamodels: a generic source metamodel called DD-IHM for Description Diagram for Human Machine Interfaces and a target metamodel called AbstractForm based on the PEAR framework, more specifically its HTML QuickForm package. Then we apply a set of generic rules to make the models operational in HMI. The first transformation performed with ATL (M2M) will transform a source model compliant with DD-IHM into a target model compliant to AbstractForm. Then, we implement M2T rules transformation using the template approach with Xpand to transform our target model into PHP code directly usable in a web application. The proposed metamodels are implemented in Eclipse with ECORE. We apply our proposal to the HMI of an online registration application.

Keywords — Interactive Systems, Model-Driven Engineering, Model Transformation, DSML, ATL, Xpand.

I. INTRODUCTION

The interactive systems modelling activity differs from conventional software engineering modelling activity in that the interactions described in the interaction models focus on the relationships between the user and the system. The development models of interactive systems must ensure the usability, i.e. the usability of a given software by a given user or category of users, or more generally, its acceptability by the users [1, 2]. The need to implement the application interfaces with the same rigor as that granted to the application itself is essential in critical systems in particular, for security reasons, and in computer applications in general for reasons of cost and usability.

Model-Driven Engineering (MDE) has led to several significant improvements in the development of complex systems by focusing on a more abstract concern than conventional programming [15]. This is a form of generative engineering in which all or part of an application is generated from models [7]. One of the key ideas is to use as many different Domain-Specific

Modeling Languages as the chronological or technological aspects of development [5, 4].

We are interested in this paper by the production of software that uses a multi-layer architecture. We focus in this paper on the presentation layer, more precisely on the automatic production of Human Machine Interface for the development of interactive Web applications. We, therefore, want to use the IDM concepts [10] to automate the production of forms.

This paper presents an automatic production approach of Human Machine Interfaces for the development of interactive Web applications. For this purpose, we, first of all, propose a metamodel (independent of all platforms) for the production of interface models independent of any platform, named DD-IHM (Description Diagram - Human Machine Interface), then we propose a metamodel called AbstractForm specific to the Framework PEAR, that will be used as a target metamodel in the overall process of developing a WEB application. In a second step, we set up a graphical editor to create the DD-IHM compliant models and then we will define a set of generic transformation rules based on the ATL language that will allow us to transform a model conforming to our metamodel (DD-IHM) to a template based on a PEAR based target metamodel, we end with the proposal of a transformation template (M2T) with Xpand. The proposed metamodels are implemented in Eclipse with ECORE. We apply our proposal to the HMI of a registration application.

This work is structured as follows: In section 2, we review the work done on HMI modelling in the interactive system development process. In section 3, we present our design approach. The source and target metamodels are presented in section 4. The target metamodel is based on the Framework PEAR. In section 5, we propose via a model the generic rules to transform the source model into a target model. Section 6 presents the physical architecture of our approach and highlights the components necessary for the implementation of an HMI according to this approach. Section 7 presents, through an example, the graphical interface of a registration request. We present a model DD-HMI, a model of AbstractForm obtained after application of the rules of transformation of a model to



Model and the rules of transformation of Model into text. We conclude in section 8 by identifying some future work.

II. LITERATURE REVIEW

Many commercial tools, often generically referred to as User Interface Management Systems (UIMS), is currently being proposed to facilitate the implementation of the interactive systems presentation layer by unspecialized programmers Combemale [2]. However, the use of such tools does not promote the automation of the entire process of developing inter-asset systems and is, for the most part, proprietary.

Still, with the goal of providing better support for user interface modelling, UMLi introduces a diagram notation for presentation layer modelling. UMLi extends the activity diagram to describe the collaboration between interaction and domain objects. In [7], the design of an HMI is seen as a series of correspondences between five metamodells: User tasks, domain concepts, workspaces, inter-actors and finally, program. The transition from one metamodel to another is done through a series of correspondences and transformations. The authors proposed a generic metamodel for the design of graphical user interfaces for mobile applications. But this metamodel does not support some interface features and does not implement events on different components.

Mohamed Lachgar presents in [13] an IDM approach to automate code generation.

Xavier Blanc presents in [4] a metamodel for the PHP language. This metamodel, designed for academic and non-industrial purposes as presented by the author, is used in an approach leading to the generation of PHP code. It only allows the generation of variables, functions and skeletons of PHP codes. This metamodel does not support design and code generation for HMIs.

In [12], the authors present a study on three metamodels based on Symbian, JavaME and .NET for the HMIs, and a contribution to the implementation of a common metamodel for these different mobile platforms. On the other hand, the authors discussed the definition of an abstract syntax for the target platforms. The fragment contained in [7] for the Java interface has the advantage that it contains almost all the elements of a GUI in Java, but it does not highlight the attributes of each class. Similarly, the authors have not provided a mechanism for managing the rules applied to the elements of a form. In the same vein, the authors of [5] propose two other metamodels: A metamodel for Symbian and a .NET Compact Framework metamodel. However, they did not present an approach for generating code from the created PSM model. In [7], authors have put in place a high-level model for the design and layout of the HMIs. They present the metamodel of the SNI in the Form of three packages: SNI UD, SNI Node and SNI Port. The SNI transformation JSP pages in Suns JSF framework automatically generate a website layout in Suns JSF framework.

Our approach in this work is similar to that described by [13, 4 and 5] for the design of a generic metamodel, PHP code generation and the automation of the construction of JSP pages in Suns JSF framework. Specifically, we will propose two metamodels. The first (DD-IHM) is platform-independent, and the second (AbstractForm) is based on the Framework PEAR globally and specifically on its package HTML_QuickForm2. We implement transformation rules (M2M) via ATL, allowing passing models from DDIHM to AbstractForm models, and we implement (M2T) rules, using Xpand transformation to obtain PHP code from AbstractForm models. All these are implemented in Eclipse for the generation of web HMI using the Framework PEAR as a gateway.

III. DESIGN APPROACH

Figure 1 presents the different design steps of our HMI approach, as we stated above.

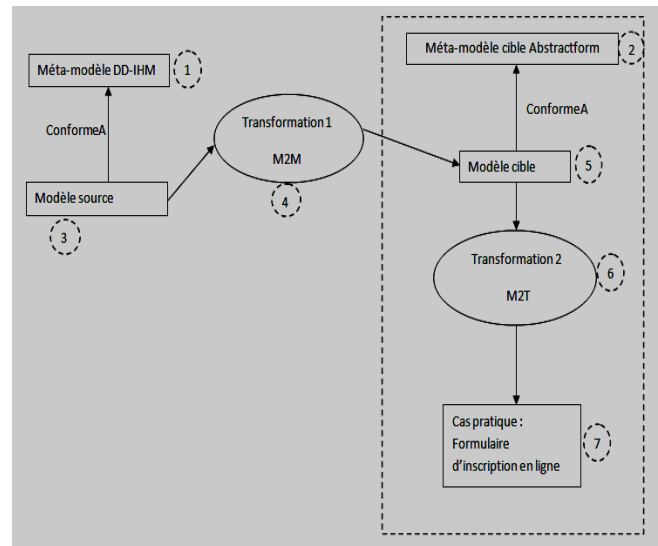


Fig. 1 The different stages of the design approach

We use the DD-IHM metamodel (1) to build a source model (3). The Model to model transformation (4) is implemented to transform a source model into a target model (5) that conforms to the AbstractForm metamodel (2). The target metamodel (5) is obtained by using the Model to Text transformation (6) to produce the final code (7).

IV. PRESENTATION OF THE METAMODELS

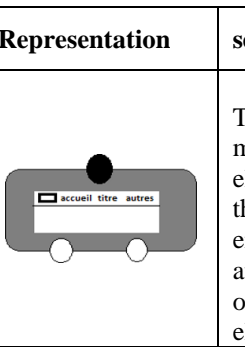
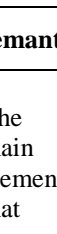
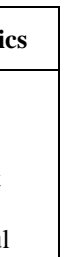
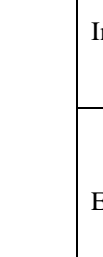
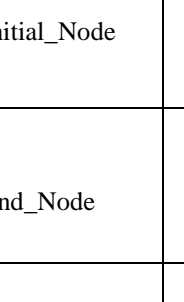
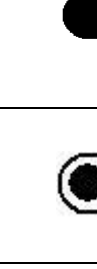
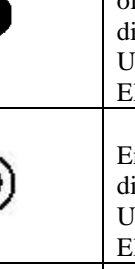
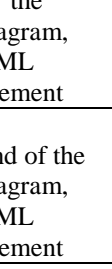
A. Source metamodel DD-IHM





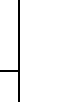
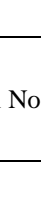
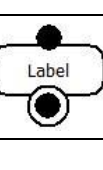
The unified modelling language (UML)[3] is an essential standard in the modelling of systems. The DD-IHM we propose is a generic metamodel based on UML and will be used to model the presentation layer of interactive systems. We use the UML metamodel class diagram and activity diagram to build new design elements. The metamodel is saved in the XML Meta-data Interchange (XMI) file. Figure 2 illustrates the DD-IHM source metamodel.

B. Graphical editor DD-IHM

The success of model-based software development is largely due to the use of a graphical editor to design models graphically. A graphical editor allows more expressions and facilities for a developer. We use the Open Source technology Sirius (based on Eclipse) to create a graphical editor which allows creating models that conform to our metamodel DD-IHM. Figure 2 represents the metamodel elements DD-IHM.

Table 1. The elements of our DD-IHM source metamodel

Element's names	Representation	semantics
Form_Class		The main element that encapsulates other elements
Group_Elt		Group element
Menu Node		
Field_Text		Allows to create of an HTML Text Field
Long_Field_Text		Allows to create of an HTML Long Text Field
Bouton		Allows to create an HTML Button
Date		Allows to create an HTML Date
CheckBox		Allows to create of an

		HTML CheckB ox
Radio		Allows to create of an HTML Radio Field
Initial_Node		Beginning of the diagram, UML Element
End_Node		End of the diagram, UML Element
Transition_No de		Materialize s the transition between two elements, UML Element
Test_Node		Use as a bridge between two or more form elements, UML Element
Synchronizatio n_Node		Materialize the synchroniz ation between the form elements, UML Element
Label Node		Use to create the label in a forme
Condition Node	[Cond]	A logical condition associated with the decision nodes

C. Description of DD-IHM :

formulas: This represents a form with its attributes. It is composed mainly of two classes: class Node and class Attribut. node is composed of three classes: Donnes, Règles and groupeELT. The class Donnes is similar

To the expression label in HTML, it allows pasting a label (Name) to an element of the Form. The class Règles 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 (groupeELT). In a group, you can have several subgroups.

Menu: As its name suggests, it allows navigating in an application. The menu can be vertical or horizontal. An HMI interface can contain several menus.

Node: It determines all the routing units in the diagram. Routing brings together the objects of the diagrams that make it possible to navigate from a menu to a form, for example. We distinguish the decision node, the merge node and the bifurcation node.

Port: A port is an abstract object attached to a node for making connections. We can associate to an entry a guard who plays the role of transitory precondition, which makes it possible to prohibit or to accept the entry in an object. In the same way, we can associate a guard that conditions the transition (postcondition) output until it is verified.

Presentation: As its name suggests, it allows to carry information on a page. The presentation can be a Message, an object, or even an object collection

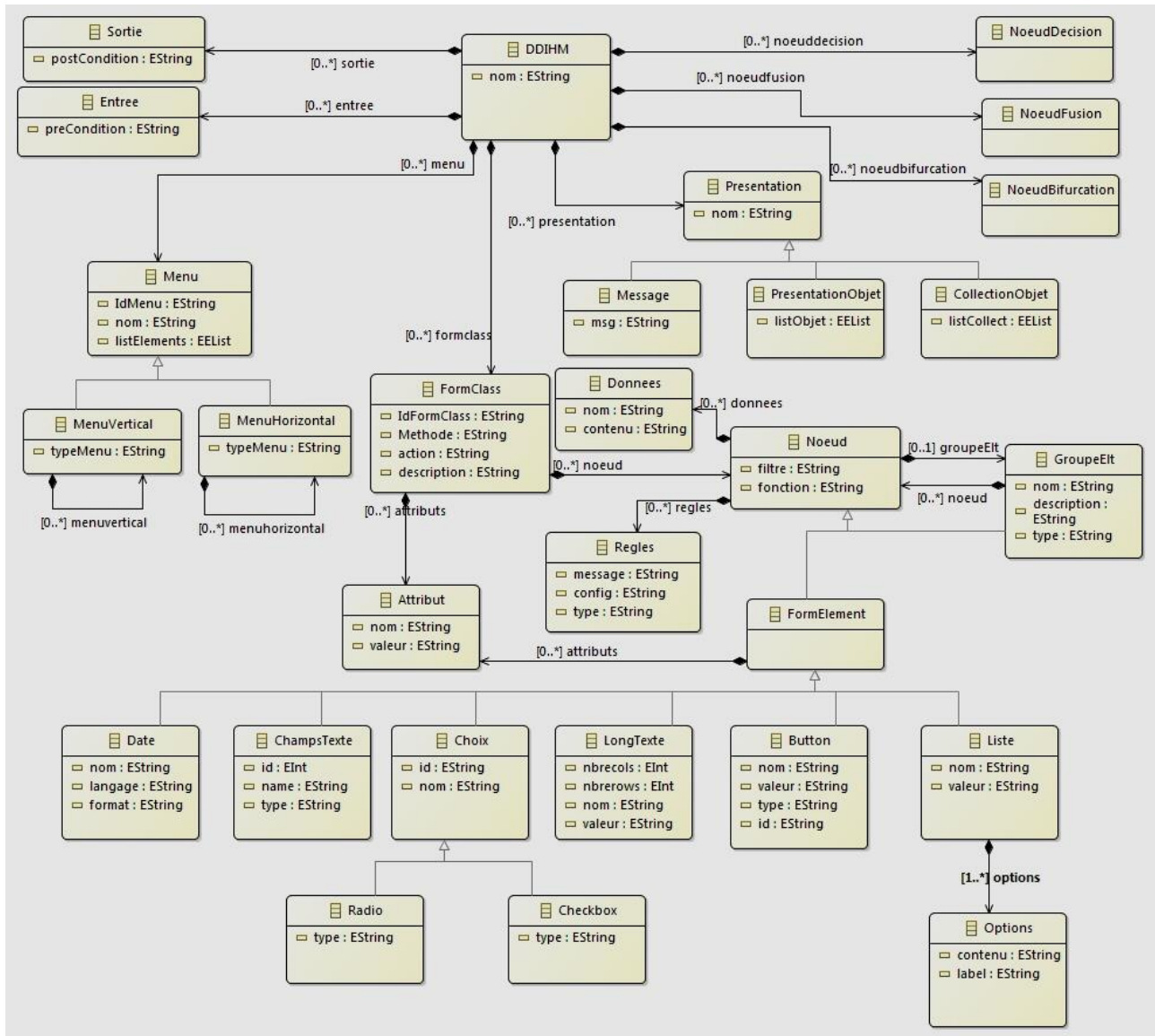


Fig. 2. DD-IHM meta-model class diagram

D. Target meta-model AbstractForm

a)The framework PEAR: PEAR (PHP Extension and Application Repository) is a collection of PHP libraries. It is a framework that allows us to manage libraries (install or update a library). Html_QuickForm is a PEAR package defining methods for creating, validating and dynamically processing HTML forms. It allows developers to make complex forms without having to deal with the HTML code. In addition, this package offers a number of features, including the validation and the control of the seizures. There are indeed other classes generating forms (oohform of phplib, for example). The advantage of the Framework PEAR is that it can be used with other packages with very powerful features like Html_QuickForm controller or DB DataObject form builder, which allows us to generate forms directly from information extracted from a database without having to write either SQL code or HTML code. We will work mainly on its second version (HTML_Quickform2), which is a rewrite in PHP5 of both Html_QuickForm and HTML_QuickForm Controller [15].

Features of HTML_QuickForm2:

- Supports all form elements defined by the HTML standard, provides several custom elements,
- Server-side and client-side validation, several common rules provided,
- Multi-page forms (tabular forms and wizards)
- Pluggable items, rules, rendering engines, and rendering plugins.

b) AbstractForm metamodel diagram: Figure 4 represents the metamodel AbstractForm it contains the set of elements allowing to build a form in conformity with HTML_QuickForm2. As can be seen in figure 4, a form consists of a set of containers and a set of elements. A container is a node that can contain other nodes.

A container is a node that can contain other nodes and a set of rules that recursively apply to its contents. An element contains an attribute, data, filter, and a set of rules.

The meta-class Model represents an HMI Model, and the meta-class Form a form of the Model.

A Model can contain one or more forms that will be spread across multiple pages and managed by a controller (meta-aggregation associations between 'Model' and 'Form'). Similarly, a form has several Node. These nodes can be containers or basic elements of a form (Meta-inheritance association of Element and Container to Node). A node contains, in addition to a set of attributes, a set of data and options necessary for the creation of the Node.

We have represented this data by the meta-class Data, containing one attribute for the Name of the data and another for the associated value. To all the nodes of the Form (text fields, buttons, checkbox, etc.), validation rules can be applied (Fields that must not be empty, a minimum number of characters, a minimum number of choices to be made, etc.). We represent this by the meta-class Rule; it contains a 'message' attribute containing the message to display if the Rule is violated and a 'config' attribute that allows configuring certain rules when the Rule is applied to a group. Hierarchical choice lists are some special types of elements represented here by the meta-class HierSelect. It is a group of several lists of choices connected to each other and whose Choice of an element of the parent's list produces the possible choice elements of the child list automatically. Unlike a simple choice list represented here by the meta-class select, the HierSelects are linked by a meta-association of aggregation to the meta-class Hoption, which represents here the options from the first list to choose from. It is the Choice made on this that triggers the loading of the possible choices in the second list represented here by the meta-class Choice; hence the aggregation meta-relationship between the two meta-classes, Hoption and Choice. A select with only one list of choices is directly linked to the meta-class Choice by an aggregation meta-association

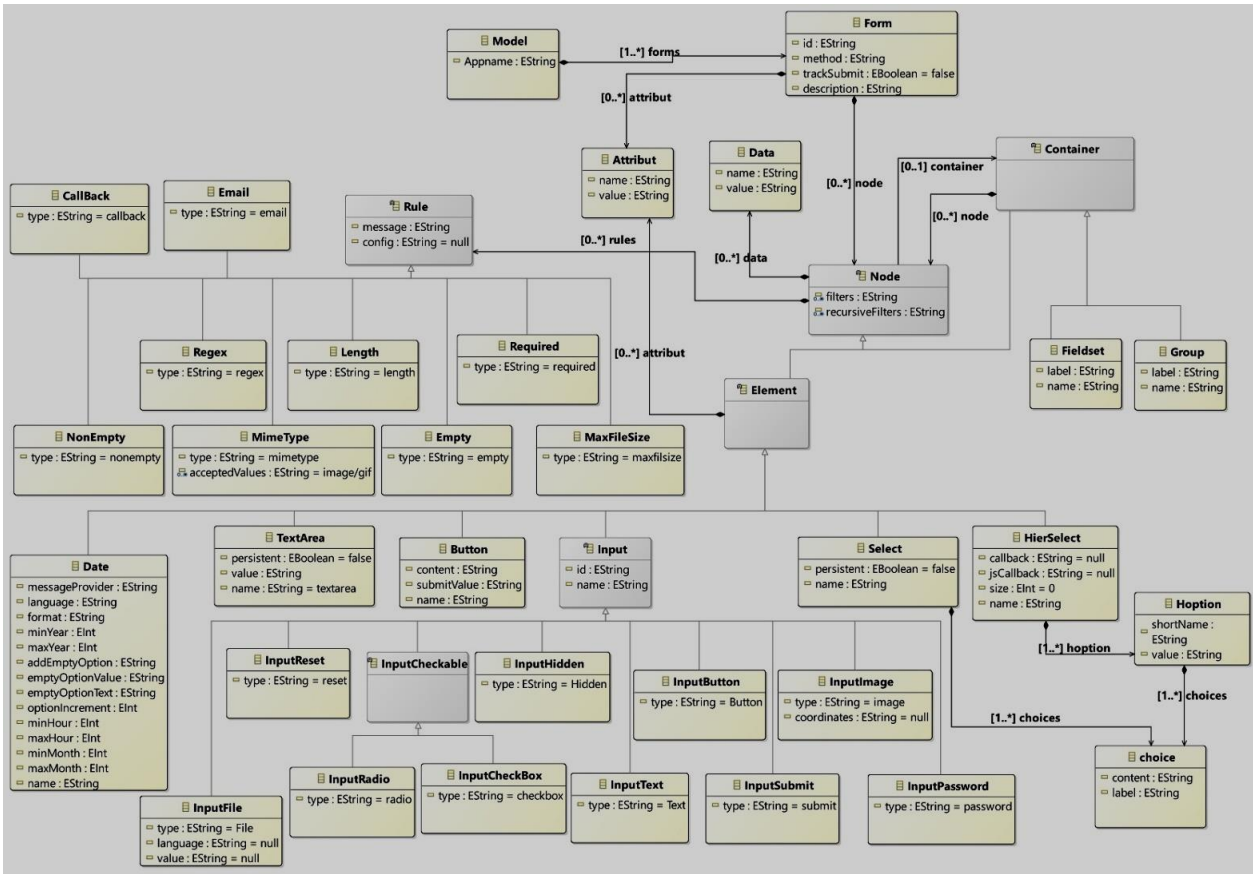


Fig. 3 Abstractform

I. TRANSFORMATIONS APPROACHES

A. External transformation: DD-IHM toward Abstract Form

For this purpose, we use ATL (Atlas transformation language), which is a model to model transformation language. In the Field of IDM, ATL provides a set of mechanisms for transforming a source model into a target model. A transformation is, therefore, an operation that takes models (source) as input and outputs models (target). That does why we first create a source model that conforms to our DD-IHM, and then we apply a set of generic rules to get a target model that conforms to the abstract Form. We will represent here an excerpt from our transformation rules:

V. TABLE II

EXTERNAL TRANSFORMATION: FROM DD-IHM TO ABSTRACT FORM

```

1 module dihm2abstractform ;
2 create OUT : abstractform from IN : ddihm ;
3
4 rule ddihm2model {
5 from dd : ddihm ! DDIHM
6 to abs : abstractform ! Model (
7 Appname <- dd.nom ,
8 forms <- dd.formclass)
9 }
10)
    
```

```

11 rule FormClass2Form {
12 from fcl : ddihm ! FormClass
13 to fo : abstractform ! Form (
14 id <- fcl.Id FormClass ,
15 method <- fcl.Methode ,
16 description <- fcl.action
17 )
18 }
19
    
```

```

20 rule Regles2Email {
21 from reg : ddihm ! Regles ( reg.type = 'email ' )
22 to rl : abstractform ! Email (
23 message <- reg.message ,
24 type <- reg.type
25 )
26 }
27
    
```

```

28 rule Rgeles2Compare {
29 from reg : ddihm ! Regles ( reg.type = 'compare' )
30 to com : abstractform ! Compare (
31 message <- reg.message ,
32 type <- reg.type
33 )
34 }
35
    
```

```

36 rule Regles2Regex {
37 from reg : ddihm ! Regles ( reg.type = 'r e g e x ' )
    
```

```

38 to regex : abstractform ! Regex (
39 message <- reg.message ,
40 type <- reg.type
41 )
42 }
43
    
```

```

44 rule Regles2Empty {
45 from reg : ddihm ! Regles ( reg.type = 'empty ' )
    
```

We will present in this section the Template conceived with Xpand containing all the rules allowing to pass from a model of AbstractForm to PHP code.

The general structure of an Xpand template: The Xpand template allows the control of code generation corresponding to a model. The Model must conform to a given metamodel [13]. The Template is stored in a file with the extension .xpt. A template file consists of one or more IMPORT statements to import meta-data-models, zero or more EXTENSIONS with the Xtend language, and one or more DEFINE blocks.

Template Xpand for AbstractForms models: Here, we present an excerpt from the Template containing the transformation rules from AbstractForm templates to PHP code.

V. TABLE III
INTERNAL TRANSFORMATION: FROM ABSTRACT FORM TO TEXT CODE

```

1  I M P O R T AbstractForm
2  D E F I N E main FOR Model
3  F I L E Appname+'.php'<
4?php require
5 once 'HTML/QuickForm2.php'; require
6 once 'HTML/head.php';
7
8 $c o n t r o l e u r = new HTML QuickForm2 Controller ( ' p r o c e s
s', false);
9  E X P A N D form FOREACH forms-
10 $c o n t r o l e u r ->run ();
11 E N D F I L E
12 E N D D E F I N E
13
14 D E F I N E form FOR Form
15 class Page this.id extends AbstractForm}
16 protected function populateForm ()
18 }
19 $this->addTabs ();
20 $form = $this->getForm ();
21 F O R E A C H node AS n
22 I F n . metaType == Group jj n . metaType == Fieldset
23 $f=$form ;
24 E X P A N D node FOR n-
25 E L S E
26 I F n . metaType == Select jj n . metaType
== Date jj n . metaType == HierSelect
27 $f=$form ;
28 E X P A N D element FOR n-
29 E L S E
30 $el = $form->addElement ( E X P A N D element
FOR n-);
31 F O R E A C H n . rules AS r
32 $el->addRule ( E X P A N D rule FOR r);
33 E N D F O R E A C H
34 E N D I F
35 E N D I F
36 E N D F O R E A C H
37 $this->addGlobalSubmit ();
38/
39
40 $c o n t r o l e u r ->addPage ( new Page this.id ( new HTML
QuickForm2 ( ' this.id' ));
41 E N D D E F I N E
42

```

```

43 D E F I N E element FOR Input
44 'this.metaType.toString().substring(16, this
.metaType.toString().length)', 'this.name', [F
O R E A C H this.attribut AS at
E X P A N D attribut FOR at- I F this.attribut != null
&& this.attribut.last() != at
, E N D I F E N D F O R E A C H ],
45 [ F O R E A C H this.data AS dt E X P A N D data
FOR dt- I F this.data != null && this.data.last() != d
t , E N D I F
E N D F O R E A C H ]
46 E N D D E F I N E
47

```

B. Internal Transformation

both placed in a folder named data, located in the PEARDIR folder.

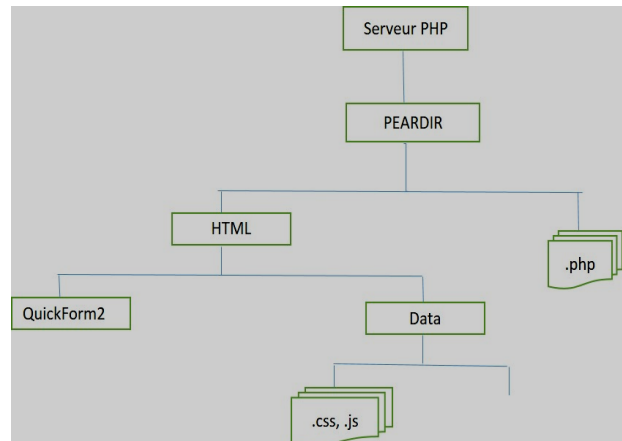


Fig. 4. Deployment architecture

The first line imports the metamodel; lines 2 to 12 define the main block of the transformation model. It contains the declaration of the output file (line 3) and the call of the production rules of the different forms of the Model (line 9). Blocks 14 to 41 define the rules used to expand a form. This is the block called on the 9th line. In this block, we also call production rules for the different elements of the Form. Some rules for producing these elements are given in lines 43-46 for the input element and 48-67 for the fieldset node.

VI. PHYSICAL MODEL

Once Pear and the package HTML QuickForm2 are installed, we will have a tree structure represented by 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 AbstractForm model is stored in the PEARDIR folder. And can therefore be run via the URL <http://localhost/peardir/registration.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),

Site registration form

Fig. 5. Registration form in our graphical editor.

VII. APPLICATION

A. Example of interface

We will design the HMI of an on ligne registration platform. The registration process will be done via a form. Anyone wishing to register will have to complete a set of fields, some of which may be required. As Field, we will have the Name, first name, sex, date of birth and spoken languages, as well as a set of rules such as regex applied to the first and last name, which only imposes certain alphabetic characters for these fields. And required set to 2 for the language group requiring the Choice of at least two languages.

B. Case Study Model

Figure 6 illustrates the drawing in our model editor of the practical case described in the preceding paragraph.

This Model conforms to our DD-IHM. The following code (table III) represents an XMI code snippet of the HIM model shown in the preceding paragraph.

VII. TABLE IV
XMI CODE OF THE SOURCE MODEL OF OUR CASE STUDY

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ddihm:DDIHM
3 xmi:version="2.0"
4 xmlns:xmi="http://www.omg.org/XMI"
5 xmlns:xsi="http://www.w3.org/2001/
6 XMLSchema-instance"
7 xmlns:ddihm="http://ddihm.com"
8 xsi:schemaLocation="http://ddihm.
9 com
10 ./MetaModel/DDIHM.ecore">
11 <form class="1" Methode="post"
12 action="InfoPersonnelles">
13 <noeud xsi:type="ddihm:GroupElt"
14 nom="infoper" description="Info
15 rmat ions
16 personnelles" type="fieldset">
17 <form element
18 xsi:type="ddihm:ChampsTexte"
19 id="1"
20 name="nom"
21 type="inputtext">
22 <regles
23 type="required"/>
24 <donnees
25 nom="label"
26 contenu="Nom"/>
27 .
28 .
29 .
30 <noeud xsi:type="ddihm:Button"
31 nom="Envoyer"
32 value="submit"/>
33 <noeud xsi:type="ddihm:Button"
34 nom="Annuler"
35 value="reset"/>
36 </form class>

```

By analyzing this code, we realize that elements form-class, TextFields, rules, Node, data ... are concepts of the source metamodel, so this Template is consistent with DD-IHM.

C. Transformation of the practical case

Transforming DD-IHM Model into AbstractForm model: The transformation of the preceding model lead to the following abstract form Model.

VII. TABLE V
XMI CODE OF THE TARGET MODEL OF OUR CASE STUDY

```

1 <?xml version="1.0" encoding="UTF-
2 8"?>
3 <abstractform:Model
4 xmi:version="2.0"
5 xmlns:xmi="http://www.omg.org/XMI"
6 xmlns:xsi="http://www.w3.org/2001/
7 XMLSchema-
8 instance"
9 xsi:schemaLocation="http://www.w3.org/2001/
10 XMLSchema-instance http://www.w3.org/2001/
11 XMLSchema-instance#xmi20010302"
12 >
13 <form class="1" Methode="post"
14 action="InfoPersonnelles">
15 <noeud xsi:type="ddihm:GroupElt"
16 nom="infoper" description="Info
17 rmat ions
18 personnelles" type="fieldset">
19 <form element
20 xsi:type="ddihm:ChampsTexte"
21 id="1"
22 name="nom"
23 type="inputtext">
24 <regles
25 type="required"/>
26 <donnees
27 nom="label"
28 contenu="Nom"/>
29 .
30 .
31 .
32 <noeud xsi:type="ddihm:Button"
33 nom="Envoyer"
34 value="submit"/>
35 <noeud xsi:type="ddihm:Button"
36 nom="Annuler"
37 value="reset"/>
38 </form class>

```

Line 10 shows the transformation result for the fieldset Node. Line 11 shows the input text for the Name present in the fieldset, the line 12 shows the 'required' Rule applied to the inputText Field. We can also see the submit and reset buttons on the 23rd and 24th lines.

AbstractForm model transformation toward PHP code: In AbstractForm, an internal transformation is done to obtain a PHP code runnable with PEAR. Bellow is the result of that transformation applied to the former XMI code.

VII. TABLE VI
RESULT OF THE INTERNAL TRANSFORMATION TOWARD PHP CODE

```

1 <?php
2 require_once 'HTML/QuickForm2.php';
3 require_once 'HTML/head.php';
4 $controleur = new HTML_QuickForm2
5 Controller(
6 process',false);
7 class Page extends AbstractForm
8 {
9 protected function populateForm()

```

Looking at this PHP code, we can see the code for the fieldset at the 12th line, the InputText for the Name (line 13) and the required Rule applied to the InputText Node at the 14th line.

D. Graphical result

The capture of the following figure 6 represents the result of the execution of the preceding code without any manual addition.

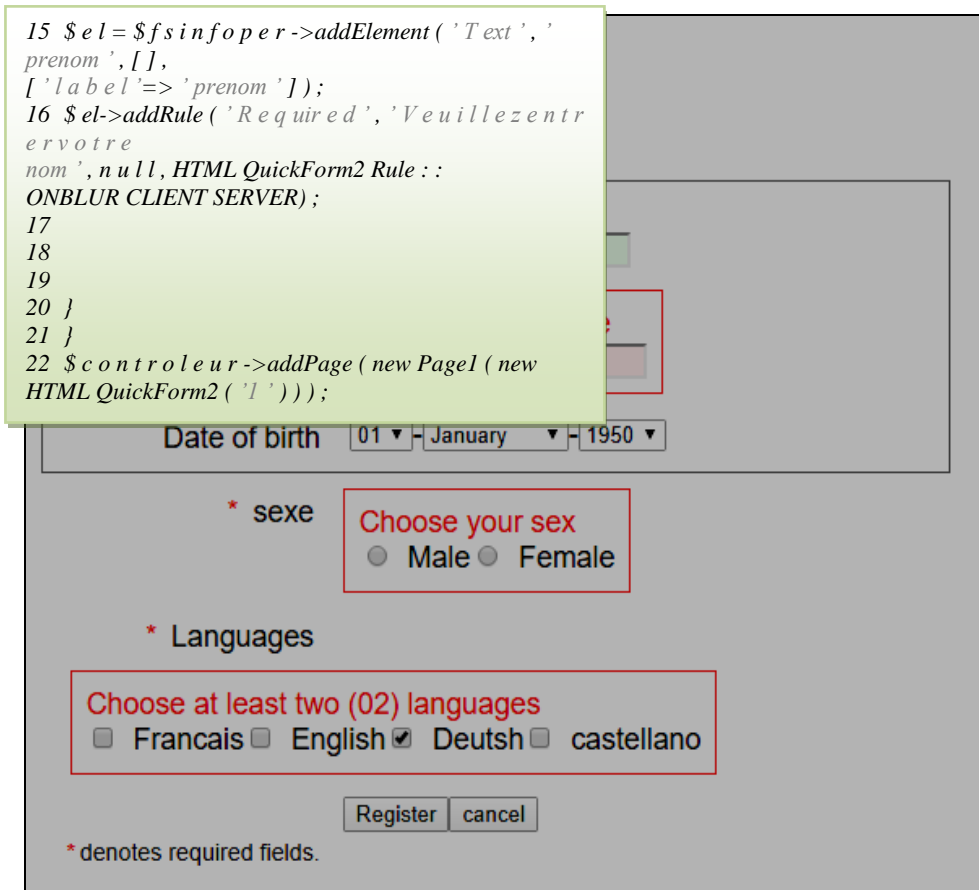


Fig. 6. Graphical result of the practical case after transformation

We can see in red the instantaneous validations of the rules that we have defined; executed on the client-side.

VIII. CONCLUSION

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's possible to increase productivity in software Engineering by using the IDM approach. For this purpose, we first propose a generic source metamodel and a target metamodel specific to the framework PEAR. Then, we set up a set of transformation rules using the ATL language, which allows us to obtain a target model from a source model. Then, from the target model generated, we use the Xpand language to transform the latter and get a PHP code. The results allow us to appreciate the gain in time and efficiency in the production of robust HMI in PHP. In the near future, we will propose a generalised approach, including the data layer and the treatment layer. This requires the extension of the metamodels so that they can take into account all the elements of the treatments layer and data layer.

REFERENCES

- [1] Etienne Andre, Christine Choppy, and Thierry Noulamo, Modelling timed concurrent systems using activity diagram patterns, Springer, *Advances in Intelligent Systems and Computing*, KSE'14, (2008) 1-15 .
- [2] BENOIT COMBEMALE, Meta-modeling approach for ´model simulation and verification: Application to process engineering, ´ PhD THESIS, Institut National Polytechnique de Toulouse, 11 (2008) 25–75
- [3] [3] Ali Koudri, Joel Champeau, Denis Aulagnier, Operational semantics for better meta-modeling, SéMo, 2007.Stein, L.D., Xavier Blanc, MDA en action, EYROLLES, Paris, ISBN 2-212-11539-3, (2008).
- [4] Jean-Bernard Crampes, Nicolas Ferry, A high-level model for the design and layout of HMIs, In: e-TI, *Electronic Review of Information Technologies*, ISSN 1114-8802, <http://www.revueeti.net/index.php/eti/article/download/29/pdf>, 5 (2008).
- [5] Paulo Pinheiro da Silva, Norman W. Paton, User Interface Modeling in UMLi, In: e-TI, *Electronic Review of Information Technologies*, In: IEEE Computer Society, <http://www.cs.man.ac.uk/norm/papers/umli.pdf>, (2008).
- [6] Jean-Sébastien Sottet, Gaëlle Calvary, Jean-Marie Favre, Engineering of Model-Driven Human-Computer Interaction, *First Days on Model-Driven Engineering, IDM'05*, Paris, (2008).
- [8] Jean Bezivin, On the Unification Power of Models, in *Software and Systems Modeling*, 4(2) (2005) 171-188. DOI: 10.1007/s10270-005-0079-0, 2004.
- [9] https://pear.php.net/package/html_quickform2 Package information : Html quickform2, https://pear.php.net/package/html_quickform2
- [10] Jean Philippe Baba, Model-driven engineering: emf modeling (eclipse modeling tools, <http://lab-sticc.univ-brest.fr/babau/cours/coursemf.pdf>, (2019).
- [11] Farah FOURATI, An idm approach of exogenous transformation from wright to ada, National School of Engineers of Sfax, (2010).
- [12] Levendovszky T. Madari L., Lengyel L., Modeling the user interface of mobile devices with dsls, In 8th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, (2007).
- [13] Mohamed Lachgar, MDA Approach to Automate Native Code Generation for Cross-Platform Mobile Applications, PhD thesis, Cadi Ayyad University (UCA); Faculty of Sciences and Techniques Guéliz (FSTG); Laboratory and institution: Applied Mathematics and Computer Science Laboratory (LAMAD), (2017).
- [14] OMG, Meta Object Facility (MOF), <http://www.omg.org/spec/QVT/1.2> qvt W3C XML Query (XQuery), <http://www.w3.org/XML/Query>, Query/View/Transformation (QVT).