American Journal of Software Engineering and Applications
2017; 6(1): 5-12
http://www.sciencepublishinggroup.com/j/ajsea
doi: 10.11648/j.ajsea.20170601.12
ISSN: 2327-2473 (Print); ISSN: 2327-249X (Online)

Towards a Platform Independent Graphical User Interface

Abdessamad Belangour*, Said Sadik, Amina Abbar

Department of Mathematics and Computer Science, Faculty of Sciences Ben M'Sik, Casablanca, Morocco

Email address:
belangour@gmail.com (A. Belangour), said.sadik.fsb@gmail.com (S. Sadik), amina.abbar@gmail.com (A. Abbar)
*Corresponding author

To cite this article:
Abdessamad Belangour, Said Sadik, Amina Abbar. Towards a Platform Independent Graphical User Interface. *American Journal of Software Engineering and Applications.* Vol. 6, No. 1, 2017, pp. 5-12. doi: 10.11648/j.ajsea.20170601.12

Received: October 29, 2016; **Accepted**: November 25, 2016; **Published**: February 24, 2017

Abstract: In classical software development processes, graphical user interfaces cannot be reused across development platforms. In addition, in MDA-based processes, they are integrated only after making the transformation of the PIM to the PSM since they belong to the target platform and hence have the same problem. They are considered part of the PSM, which deprives us from reusing them as we do for the business logic. In this paper, we aim at proposing a common platform independent graphical user interface library that represents the presentation logic in terms of inputs and outputs. This is achievable through proposing a generic metamodel for basic GUI controls that focus on getting and presenting data rather than those of ergonomic purposes. This metamodel will enable us to build generic graphical interfaces that can be transformed to any of the market libraries such as AWT, SWING, WinForms, Tkinter. That is why we built metamodels for those libraries and defined mappings between the generic metamodel and those libraries metamodels. Finally, the generic GUI library is used to make PIM-GUIs that are kept with business-PIMs and that can together be reused in a way that is independent from any development platform. Final mappings transforms these PIM-GUIs into platform bound GUIs or PSM-GUIs such those we mentioned earlier or any future graphical library.

Keywords: Model Driven Architecture, Model Driven Engineering, Graphical User Interface, Metamodel, Generic, Platform Independent

1. Introduction

The OMG (Object Management Group) has defined the Model Driven Architecture or MDA™ [1] as part of its response to the increasing complexity, heterogeneity and evolutivity issues of information systems.

It tackled these issues through the rising of the level of abstraction by adopting models instead of objects as a first measure and the separation of the business logic of an information system from the implementation of that logic on a specific technological platform (CORBA, C#/DotNet, Java/EJB, XML/SOAP, etc.) as a second one. Thus, the simple principle of MDA is the elaboration of platform independent models (known as PIMs) and their transformation into platform specific models for a given platform (known as PSMs). The techniques used are essentially modeling techniques and model transformation techniques [2].

While most of the proposed approaches focused on extracting logic business in form of PIMs, as the logic business is the most important part of an information, system, presentation logic has been ignored to a certain extent and thus their elaboration still belongs to the post-PSM stage. As a direct consequence, a waste of effort is made to rewrite code for graphical user interfaces when shifting platforms even though this shift is made in scale of many years. Instead, it would be simple to have those GUIs also as PIMs and to automatically generate PSM-GUIs for new platforms.

To achieve this goal, we propose in this paper a metamodel that represents a platform Independent Graphical User Interface library that can be used to make PIM-GUIs as well as a transformation tool that transforms them into known PSM-GUIs such as Java AWT and SWING, .Net Winforms and Python Tkinter.

2. A Platform Independent Model GUI Library

2.1. Elements to Capture

Common applications interact with end-user through Graphical User Interfaces. They rely on these GUI interfaces to get or to present information to the end user. While these GUI interfaces can differ in design and richness of graphical widgets they contain, basic input and output widgets remain the same. Therefore, in the various existing platform-bound GUI libraries on the market, graphical elements such as textboxes, checkboxes, buttons, etc., represent the same concepts regardless of the names given to these elements in different technological frameworks. Thus, for example, a text box is called a TextField in Java AWT library, JTextField in Java SWING or simply Textbox in. Net WinForms, etc... Classes whose members (properties and methods) deliver similar functionality although having quite different names and parameters represent these graphical elements.

To build a platform independent graphical user interface library, basic common graphical elements across platform-bound GUI libraries need to be examined in terms of inputs or outputs rather that of ergonomic purposes. The purpose is to cross those GUIs from the PSM space to the PIM space. Consequently, an application can have a quite complete platform independent description and guaranteed against technological obsolescence.

2.2. Proposition of a GUI-Independent Metamodel

After studying different GUI libraries, we were able to identify the following basic platform independent graphical user interface elements: Frame, Panel, GroupBox, TextField, Button, ListBox, RadioButton, ComboBox, Image, Label, and TextArea.

3. A Generic Metamodel

In the interest of making PIM to PSM transformations possible, we went through many PSM graphical user interface frameworks: precisely the very code used to create the graphical user interface, some of which are AWT, SWING, .NET WinForms and Tkinter so as to create an abstract metamodel which is framework free. This metamodel can be changed in any given time to include new generic basic components we might find later on.

We were able to extract the most common I/O components, such as TextField and Button. These components are mainly divided into two sections: Containers and widgets. Containers are the main UI components in which widgets are put together to form a single window while widgets are controls used to collect or present information to the user. Menus are also included as a mean to navigate the offered business functionality.

There are, however, some controls found only in certain, if not in only one framework, such as the DataGrid in Winforms, for this specific component for example, a custom transformation is possible, if we take Java SWING for instance, we can create a JTable, include some hyperlinks, and the necessary commands for each hyperlink (the CRUD operations in the DataGrid component).

The metamodel relied on Composite Design Pattern to avoid unnecessary compositions and thus having a simple, clean metamodel.

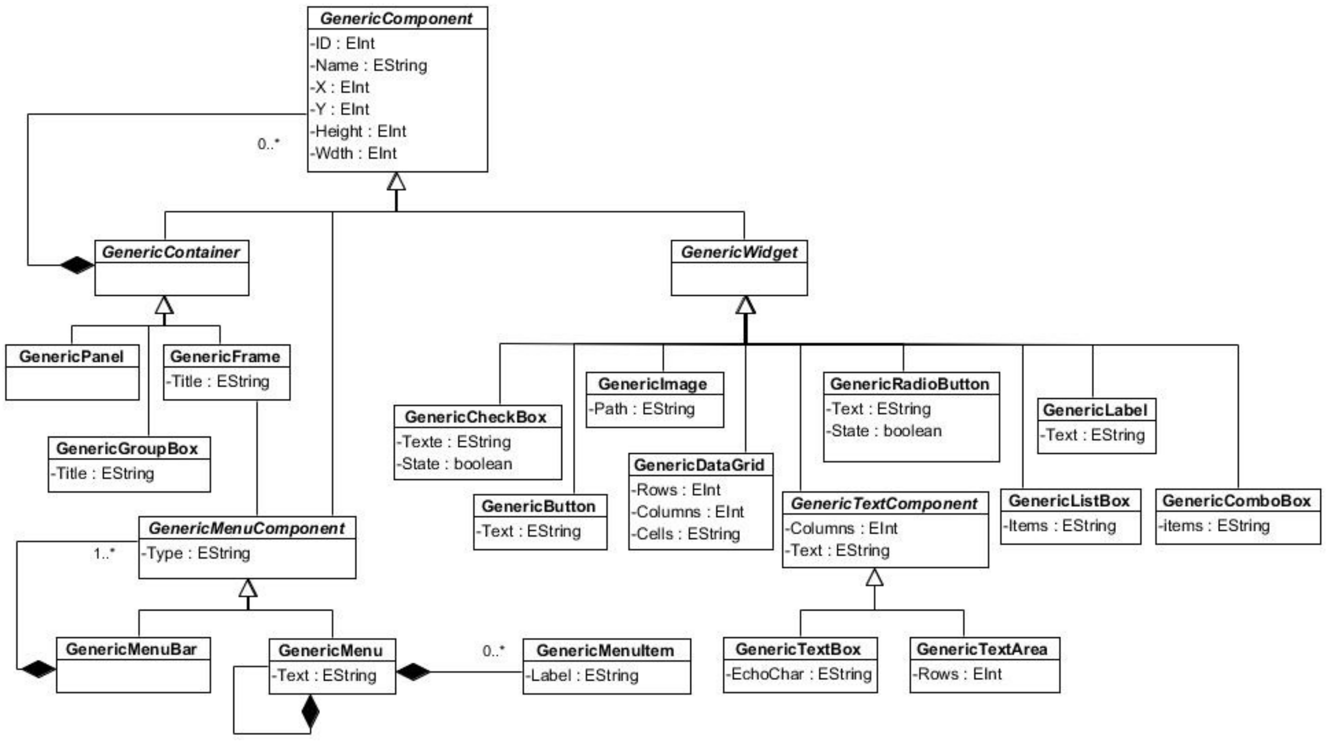

Fig. 1. *Generic Metamodel.*

4. A Metamodel for Java AWT Library

4.1. Description of the AWT Library

AWT [3] stands for Abstract Window Toolkit. It is graphical user interface library for the Java SE platform. AWT was Java's first GUI framework [8], and it was included in Java since version 1.0. It allows the user to create windows and simple controls through its various classes and methods. AWT contains a set of classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a Textbox is known as a component in AWT terminology. There are mainly to kind of components: containers and contained components. A container is a component that can contain components and other containers. Examples of containers are Frame and Window classes. Examples of contained components are Button and TextField …etc.

4.2. AWT Metamodel

Oracle stores its AWT Graphical user interface library in a file named "rt. jar". As a first step and in order to match the exact classes in AWT we used reverse-engineering to get its complete hierarchy. As a second step, we got rid of all components that are not basic or are not part of graphical components. Finally, we build our metamodel with metaclasses names prefixed with AWT prefix to distinguish it from other graphical metamodels.

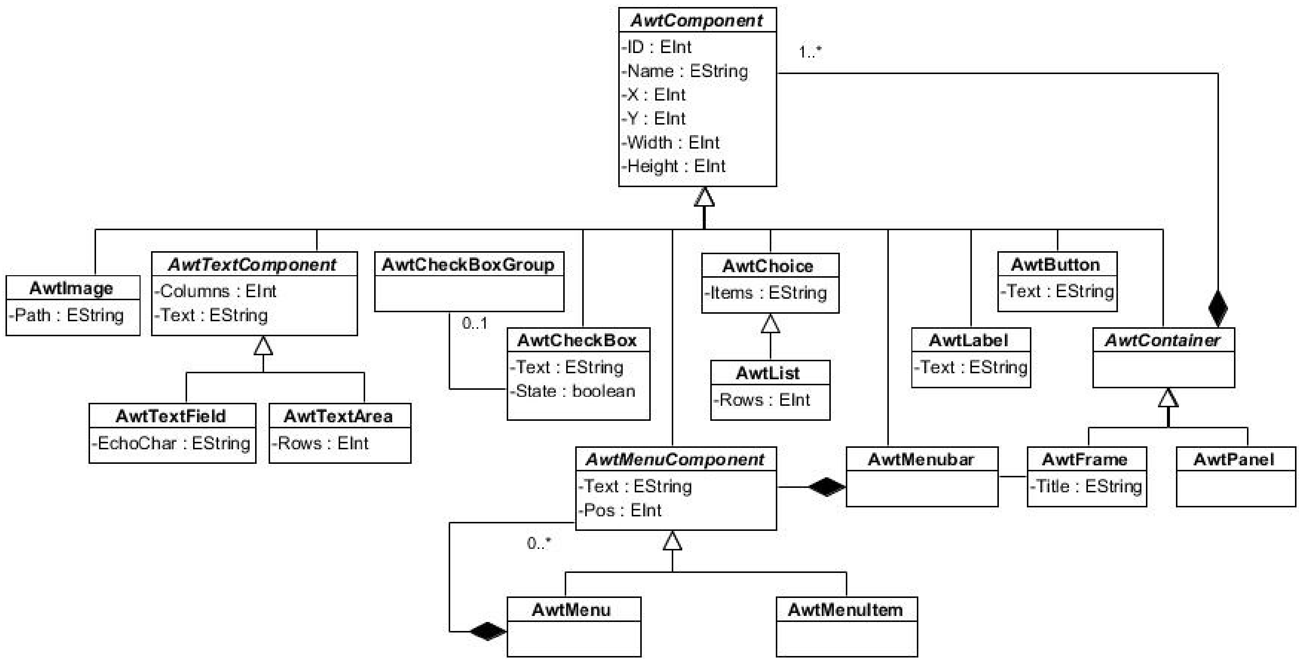

Fig. 1. *AWT Metamodel.*

5. A Metamodel for Java Swing Library

5.1. Description of the SWING Library

Java Swing [4, 14] is a Java Graphical User interface Widget toolkit which contains a rich set of components and widgets to build a GUI with a native look but still having the advantage of platform independence (meaning it can run similarly in every OS).

Swing is the response to AWT problem [8], the use of the native peers for each component or container, as a result, the UI varies depending on the operating system, and it may even act differently. AWT can also run its bugs [10] more often because its peer-based approach relies heavily on the underlying platform.

Swing includes built-in controls such as trees, image buttons, tabbed panes, sliders, toolbars, color choosers, tables, etc. Swing components are written entirely in Java and thus are platform-independent. It provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms.

5.2. SWING Metamodel

Just like the previous metamodel, SWING classes (which would eventually become metaclasses since we are creating a metamodel) were found in the same rt. jar, precisely under "javax".

Much like the AWT metamodel, SWING metamodel contains the widely used containers and components, some of which are already present in AWT, normal since SWING inherits from the original graphical user interface library, but has more sophisticated controls than AWT, which in turn can be made through combining some of AWT components.

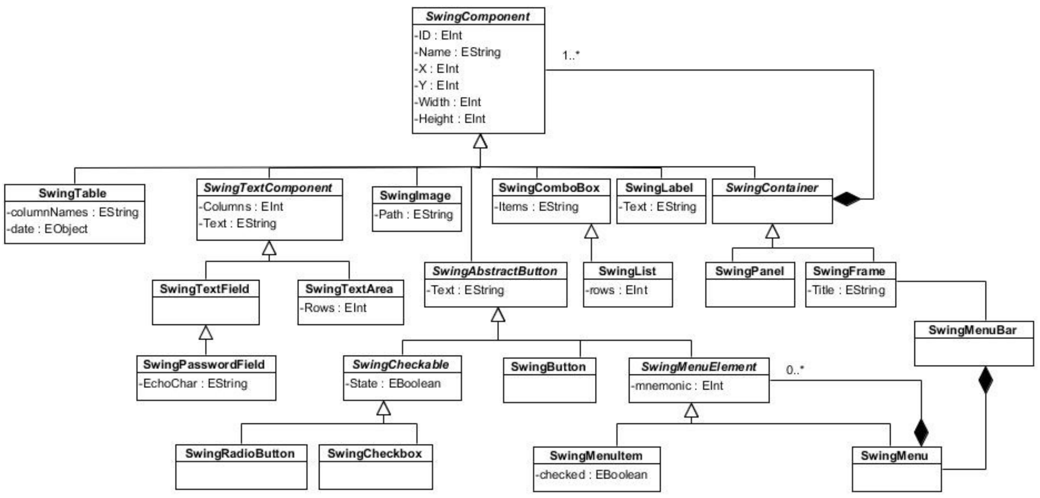

Fig. 3. *SWING Metamodel.*

6. A Metamodel for DotNet Winforms Library

6.1. Description of the DotNet Winforms Library

Windows Forms [5] is a smart client technology for the. NET Framework. It is a set of libraries that enables creating graphical user interfaces with ease, compared to AWT and SWING since the IDE (Visual Studio) gives a quick and simple way to create GUIs using drag and drop –of course, we can still do that for Java in NetBeans but it is much easier in Visual Studio.

Since it is a windows product, it can take advantage of the windows authentication [12], to keep things simple and straightforward, using Windows login remains the best way to authorize a user.

In Windows Forms, a form is a visual surface on which information is displayed to the user. Windows Forms applications are built by adding controls to forms and developing responses to user actions, such as mouse clicks or key presses. A control is a discrete user interface (UI) element that displays data or accepts data input.

6.2. DotNet Winforms Metamodel

DotNet Winforms is the equivalent of AWT or SWING for Windows applications, it has a rich set of graphical user interface components, which presents a real challenge to retrieve only the mainly used ones, of course after carefully studying the previous frameworks and this one as well, the abstract metamodel served as a reference point for the components.

Thanks to reverse engineering, the exact hierarchy is shown for each and every element present in the metamodel, customized into the composite pattern for a more elegant display.

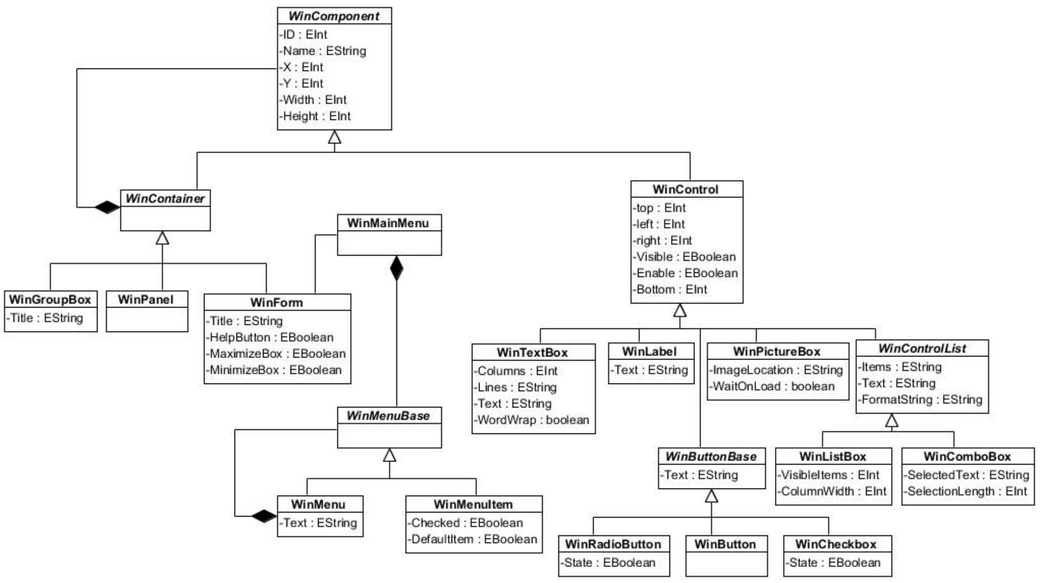

Fig. 4. *Winforms Metamodel.*

7. A Metamodel for Python Library

7.1. Description of the Python Library

Unlike the previous languages (C# and Java), Python [16, 17] is a high level programming language. Its less verbose and utterly simple syntax is a powerful asset to create clear programs on both large and small scales.

Tkinter comes with most Python distributions [16], it provides an object-oriented layer on top of the Tcl/Tk GUI library and runs on Windows, UNIX, and Macintosh systems.

Even though Tkinter does not provide a large set of controls, it does come with extension libraries to expend its capabilities, and it enables the users to use advanced GUI controls, and create customizable ones.

Python has many GUI libraries, since Tkinter provides the basic ones, such as wxPython [18], AVC, formLayout etc.

7.2. Tkinter Metamodel

Python is popular among programmers, for its many qualities that other languages such as Java and C# lack, such as the fast develop/debug. For instance, most of the internet protocols are written in Python, etc. For these facts alone, having a means to easily convert GUI to Python is indispensable. The following metamodel Fig. 5 shows the basic controls that the previous libraries have included in their metamodels.

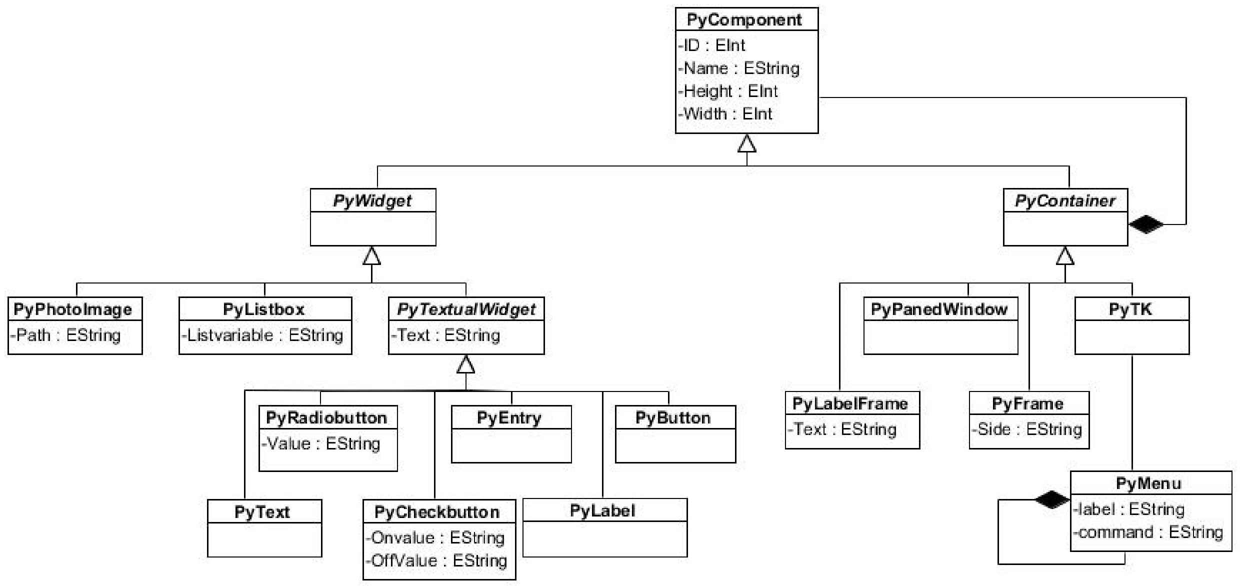

Fig. 5. *Tkinter Metamodel.*

8. Transforming a PIM GUI Interface to a PSM GUI Interface

8.1. Transforming a PIM GUI Interface to AWT

After examining the various present metamodels, transformation cannot be easier, since most components are easily recognizable as in Java AWT for instance. Consequently, we can easily go from the abstract element to a platform-specific graphical user interface component, resulting in a smooth transformation process.

Unfortunately, that is generally not the case for the rest of the frameworks. Names do change from a technology to another; ergo there is a need to make a list of generic elements and their correspondence in the platform-specific GUIs (see Table 1).

Table 1. *Generic to AWT mappings.*

PIM GUI	JAVA-AWT
GenericFrame	AwtFrame
GenericPanel	AwtPanel
GenericGroupBox	AwtLabel
GenericLabel	AwtLabel
GenericTextField	AwtTextField
GenericTextArea	AwtTextArea
GenericButton	AwtButton
GenericCheckBox	AwtCheckBox
GenericRadioButton	AwtCheckBox+AwtCheckBoxGroup
GenericListBox	AwtList
GenericComboBox	AwtChoice
GenericDataGrid	A grid of AwtLabel
GenericImage	AwtImage
GenericMenuBar	AwtMenuBar
GenericMenu	AwtMenu
GenericMenuItem	AwtMenuItem

8.2. Transforming a PIM GUI Interface to SWING

Although AWT and SWING share the same technology, which is Java, they cannot be more different, since Swing is an evolved technology, components that do not exist in AWT are found. That is why it is imperative to have the changes listed (see Table 2).

Table 2. *Generic to SWING mappings.*

PIM GUI	JAVA-SWING
GenericFrame	SwingFrame
GenericPanel	SwingPanel
GenericGroupBox	SwingLabel
GenericLabel	SwingLabel
GenericTextField	SwingTextField
GenericTextArea	SwingTextArea
GenericButton	SwingButton
GenericCheckBox	SwingCheckBox
GenericRadioButton	SwingRadioButton
GenericListBox	SwingList
GenericComboBox	SwingComboBox
GenericDataGrid	SwingTable
GenericImage	SwingImage
GenericMenuBar	SwingMenuBar
GenericMenu	SwingMenu
GenericMenuItem	SwingMenuItem

8.3. Transforming a PIM GUI Interface to WINFORMS

Microsoft has various graphical user interface libraries including Winforms. Winforms of course has components that are in common with previously seen frameworks, but components names do differ. Listed below is the different components stated in Metamodel Fig. 4, as well as generic elements and their peers in. Net Winforms.

Table 3. *Generic to WINFORMS mappings.*

PIM GUI	WINFORMS
GenericFrame	WinForm
GenericPanel	WinPanel
GenericGroupBox	WinGroupBox
GenericLabel	WinLabel
GenericTextField	WinTextBox
GenericTextArea	WinTextBox
GenericButton	WinButton
GenericCheckBox	WinCheckBox
GenericRadioButton	WinRadioButton
GenericListBox	WinListBox
GenericComboBox	WinComboBox
GenericDataGrid	WinDataGrid
GenericImage	WinPictureBox
GenericMenuBar	WinMainMenu
GenericMenu	WinMenu
GenericMenuItem	WinMenuItem

8.4. Transforming a PIM GUI Interface to Tkinter Python

Python has many GUI libraries, but Tkinter remains the de facto standard GUI. Since we are looking for simplicity and mostly interoperability, and Tkinter is included in Windows and Mac's versions of python, it is the best choice for the transformation engine, but there are controls found in other libraries as well, such as wxPython for the sophisticated Datagrid. Here is the mapping from the Generic controls to Tkinter's controls.

Table 4. *Generic to Tkinter mappings.*

PIM GUI	Tkinter Python
GenericFrame	PyTk
GenericPanel	PyFrame
GenericGroupBox	PyLabelFrame
GenericLabel	PyLabel
GenericTextField	PyEntry
GenericTextArea	PyText
GenericButton	PyButton
GenericCheckBox	PyCheckbutton
GenericRadioButton	PyRadiobutton
GenericListBox	PyListbox
GenericComboBox	PyListbox
GenericDataGrid	A set of PyLabels
GenericImage	PyPhotoImage
GenericMenuBar	PyMenu
GenericMenu	PyMenu
GenericMenuItem	PyMenu

9. Atlas Transformation Language ATL

9.1. Definition

ATL (Atlas Transformation Language) is a model transformation language created by the Atlas group (INRIA & LINA) [6]. Its relatively simple syntax allows quicker and easier transformations from a source model to a target model. It is widely used in the field of MDA (Model Driven Architecture) and MDE (Model Driven Engineering). It may be an Eclipse add-in, but the libraries can be used to develop richer transformation applications, such as the transformation engine, which is described in this paper.

MDA [9, 15] is an approach to software design development and implementation, it uses models to structure design specifications.

An ATL transformation file has two sections, one is mandatory, the header section where we declare the source and the target models, as well as the helpers and the transformation rules.

9.2. Structure of Transformations

Transformation rules [7] are the most fundamental components in a transformation file used to express the transformation process.

ATL rules are composed of a source pattern and of a target pattern. The source pattern as well as the target pattern are respectively composed of a set of elements that specify a type in the source and target metamodel. In both patterns, a type from the source metamodel is selected and then related to another type in the target metamodel. Additionally the target type's elements are bound to their equivalents in the source type.

10. Transformation Engine

In the previous sections, we defined a generic GUI metamodel as well as metamodels for AWT, SWING, TKinter and Winforms libraries. Mappings were then elaborated between the GUI metamodel elements form one side and the specific library elements from another side. In this section, we show how we built an engine that execute the specified transformations according to a two steps process. The first step is to build a generic GUI model (serialized in XMI) that conforms to the genericGUI metamodel as an input for the engine. XMI (XML Metadata Interchange) [11] is a use of the Extensible Markup Language (XML) that is intended to provide a standard way for programmers and other users to exchange information about metadata.

After that, the user chooses a specific library as a target. This causes it is metamodel to be loaded in the engine and according to the predefined mappings model elements to be created. The output model is an XMI file that represents a model, which conforms to the target library (Fig. 6).

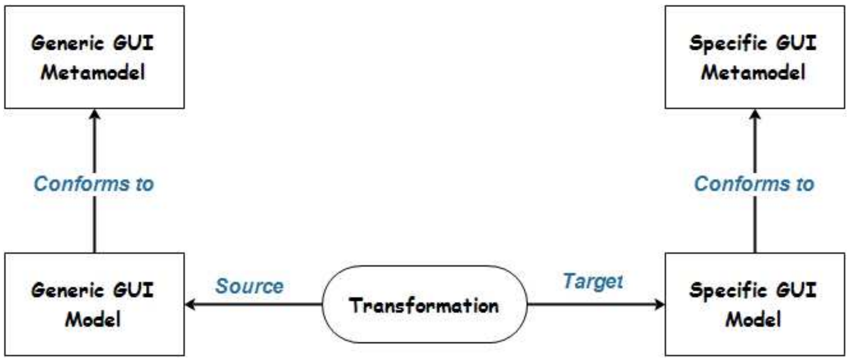

Fig. 6. *ATL transformation process.*

The second step of the transformation transforms the output XMI file to Code according to the specific library development platform. Thus, for AWT and SWING code is Java, for Winforms code is C# and for Tkinter code is Python as we chose for our engine.

The transformation engine is a set of tasks. Firstly, we created ecore files which are metamodels representations in ATL terminology. Thus, five ecore files were created: the source metamodel, which is the generic ecore, and the target metamodels (AWT, Swing, Tkinter and Winforms). After that, the source model was instantiated from the abstract metamodel. For each target metamodel, an ATL transformation file is needed: thus, we have four ATL files. The EMF plugin allows us to create the implementation of the ATL transformation files, which provides the necessary modules for the automated transformation. Once the target model is specified, the transformation engine creates the target model, based on the corresponding ATL file.

EMF [13] (Eclipse modeling framework) being part of MDA, is the implementation of a light version of MDA in Eclipse.

The target model, which is an XMI file, is read and parsed to extract the graphical components. Once the components are extracted, the transformation engine takes the data and builds the graphical user interfaces accordingly. As mentioned earlier, if it is AWT or SWING, then it is done in Java. If it is Winforms then a C# application is called, and a web service is started causing the data to be transferred to the C# application that reads all the components and builds the GUI, otherwise it calls the corresponding executable for the chosen language, calls the web services and builds the GUI.

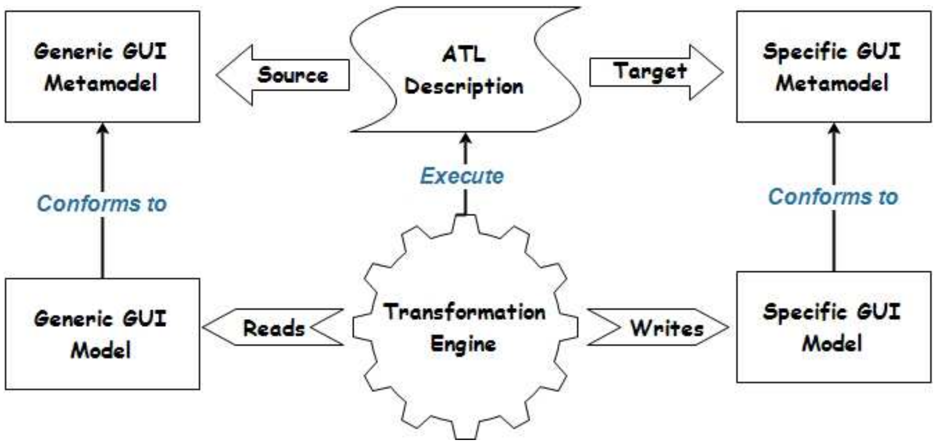

Fig. 7. *Transformation engine process.*

11. Conclusion

In this paper, we presented an approach of isolating the presentation logic of GUI applications from the implementation space according to the MDA initiative. Thus, not only the business logic of an application is preserved across technologies, but also its graphical user interface. This goal was firstly achieved by proposing an abstract generic metamodel of basic input output graphical controls. We then, proposed a metamodel for every targeted GUI library. Finally, Automatic transformations, we realized through a tool developed for this purpose to generate the corresponding code.

References

[1] Object Management Group: the MDA Guide Version 1.0.1, document OMG/2003-06-01, (2003).

[2] Jan Oyvind Aagedal, Jean Bézivin., Peter F. Linington: “Model-Driven Development”, ECOOP 2004 Workshop, Oslo, Norway, (2004).

[3] Oracle Help Center: Java Platform Standard Edition 7 Documentation, Java AWT.

[4] Oracle Help Center: Java Platform Standard Edition 7 Documentation, javax. Swing.

[5] Microsoft: MSDN, Windows Forms Overview.

[6] ATLAS group, LINA & INRIA: ATL: Atlas Transformation Language, User Manual, Nantes, January 2005, (2005).

[7] Frédéric Jouault, Ivan Kurtev: Transforming Models with ATL, Nantes. MoDELS 2005 International Workshops Doctoral Symposium, Educators Symposium Montego Bay, Jamaica, October 2-7, 2005 Revised Selected Papers. pp 128-138, (2006).

[8] Herbert Schildt: Java the complete reference 9th edition, comprehensive coverage of the Java language, (2014).

[9] Wiley Publishing, Inc. David S. Frankel: Model Driven Architecture, Applying MDA to Enterprise Computing, (2008).

[10] Daniel Liang, Introduction to Java programming, 6th edition, (2007).

[11] Stephen Brodsky, Object Interchange with XMI, June 2000, (2000).

[12] Bill Sempf, Chuck Sphar, Stephen Randy Davis: C# 5.0 ALL-IN-ONE FOR Dummies, (2013).

[13] Bill Moore, David Dean, Anna Gerber, Gunnar Wagenknecht, Philippe Vanderheyden: Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, February 2004, (2004).

[14] Chet Haase, Romain Guy: Filthy Rich Clients, Developing Animated and Graphical Effects for Desktop Java™ Applications, (2007).

[15] Cephas Consulting Corp: The fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture (MDA), January 2006 (2006).

[16] Alex Martelli, Anna Matelli Revenscroft, David Ascher: Python Cookbook, 2end edition, March 2005 (2005).

[17] Mark Lutz, Python Pocket Reference, 4th Edition: Python in Your Pocket, (2009).

[18] Bhaskar Chaudhary, Tkinter GUI Application Development Blueprints: Master GUI programming in Tkinter as you design, implement, and deliver ten real-world applications from start to finish (2015).