Generating Graphic User Interface of Web Applications Using Source Code Generator Based on Dynamic Frames

Danijel Radošević, Denis Bračun, Ivan Magdalenić

University of Zagreb, Faculty of Organization and Informatics, Varazdin 42000, Croatia

Received: ______________ / Accepted: ______________ / Published: ______________

Abstract: This paper deals with a problem of application generation together with their Graphic User Interface (GUI). Particularly, the source code generator based on dynamic frames was improved for more effective specification of GUI. It’s too demanding for the developers to have specification of the application that contain all physical coordinates and other details of buttons and other GUI elements. The developed solution for this problem is based on post-processing of generated source code using iterators for specifying coordinates and other values of graphic elements. The paper includes two examples of generating web applications and their GUI.

Key words: generators, Graphic User Interface, dynamic frames

1. Introduction

Dynamic frames generator model, introduced in [10] is a model of source code generator that we use for building of complete applications following principles of Software Product Lines (SPL). It has some similarities with other SPL approaches, including frames techniques (e.g. XVCL [4]), Generative Programming [2], Aspect Oriented Programming [5] and approaches based on scripting languages (e.g. Open Promol [13] and CodeWorker [6]). Unlike some other approaches, our model clearly separates three basic components: Specification (S) that describes application characteristics, Configuration (C) that describes the rules for building applications, and Templates (T) that refer to application building blocks.

Different features of application to be generated can be easily specified in Specification, as values of appropriate attributes. The arrangement of elements on GUI could be also specified by attribute-value pairs, but that could be demanding, making Specification bulky. For this reason, generator model was improved for more effective specification of GUI. The solution based on post-processing of generated code is implemented in two examples of web applications generation. In the first example, the arrangement of GUI elements is defined using HTML layers, while the second example generates XAML document to define the arrangement of GUI elements of Silverlight based application.

Section 2 gives presentation of related work, Section 3 introduces Dynamic frames generator model, Section 4 presents the given solution for generation of
GUI, Section 5 gives an example of generating XAML document and Section 6 is conclusion.

2. Related work

The most comparable approach to our Dynamic frames generator model [10] is Jarzabek's Xml-based Variant Configuration Language (XVCL) [4]. XVCL is a variant mechanism that uses x-frames to define building blocks of source code to be generated. All used x-frames make a tree structure, as shown in Fig. 1.

Fig 1. XVCL x-frames [3]

XVCL distinguish specification x-frames that contain program specification, while other x-frames combine program code with break sections to define insertion of variable program parts. Configuration elements are specified implicitly, in break sections, defining different kinds of insertion and adaptation. As described in Section 3, Dynamic frames generator model also define a tree structure, but developer has to define only top level frames, while other frames are instantiated dynamically, during the process of generation. Also, each frame contains clearly separated parts regarding to Specification, Configuration and code template (particular template from Templates).

Automatic generation of GUI could be integrated with some other features of target applications, like security. Schlapfer [12] has combined GUI functionality with awareness of the security policy. In traditional approach, GUI elements that are not allowed for the particular user are shown in gray, as shown in Fig. 2.

Fig 2: An example window. a) view of HR members; b) view of other employees [12]

The role of generators, as described by Schlapfer [12] is to generate the GUI without redundant elements, i.e. appropriate to particular user. So, there is a problem of arranging GUI elements, and specification of that arrangement. Schlapfer provided a method to validate the definition of smart, security-aware GUIs by checking the corresponding OCL expressions over the SecureUML+GUI models that capture the smart, security aware GUI definitions [12].

Different XML variants are used for defining of GUI layouts. XF (Extensible Formatting language) is a generic high-level formatting language serving as a mediator for contemporary, powerful formatting languages such as CSS and XSL [14]. XUL is the model used by the Mozilla family of browsers and has a rich notation for creating widgets, and uses Box, Grid and other layout models [1]. EXtensible Application Markup Language (XAML) is a user interface mark-up language for Windows Presentation Foundation (WPF) and consists of features from both Microsoft Windows applications and web applications [7]. There are some other examples XML variants used for defining of GUI layouts, like XForms and LZX [7]. It’s important that these XML, HTML and similar documents can be easily generated using our Dynamic frames generator model, as shown in given examples (Section 4 and 5).

3. Dynamic Frames Generator Model

Dynamic frames generator model [10] was developed on a basis of previously introduced Scripting Generator Model (SGM) [8]. The main purpose of SGM was to
offer a model for modeling of generators written in scripting languages like Perl and PHP. SGM separates program specification and set of code templates used as building blocks of programs to be generated, but configuration was inlined in a generator itself. That was changed in the Dynamic frames model, that defines generator from three kinds of elements: Specification (S), Configuration (C) and Templates (T). All three model elements together make the SCT frame (Fig. 3):

![Fig. 3: SCT frame [9]](image)

**Specification** contains attribute-value pairs that define features of generated application. **Template** is a set of all code templates. Each code template contains source code in target programming language together with connections (replacing marks for insertion of variable code parts). **Configuration** defines the connection rules between Specification and Template in a form of triplets (connection-attribute-template). Connections are replacing marks (usually enclosed by special signs ‘#’) in code templates. During the process of generation, connections are replaced by another code templates and/or values from Specification.

Starting SCT frame\(^1\) contains the whole Specification, the whole Configuration, but only the base template from the set of all Templates. Other SCT frames are produced dynamically, for each connection in template, forming generation tree (Fig. 4).

Each frame produces one fragment of source code in process of program generation. The final program code is built from all fragments of source code by source code generator.

![Fig. 4: The generation tree [9]](image)

### 3.1. Handler

Handler makes generator scalable, enabling generation of more target program files from same Specification. In other words, Handler prepares inputs for the generator and then collects and saves generator outputs (Fig. 5).

As shown in Fig. 5, Handler uses initial lines in Configuration to define top-level code templates, as bases for building of appropriate generation trees. This lines of Configuration are connected to appropriate lines in Specification (contain **OUTPUT** attribute), defining kinds of outputs. Each kind of output can be used in generation of more output files (e.g. **output** is used in generation of **output/students.cgi, output/courses.cgi, output/exams.cgi** and **output/questionnaire.cgi**, as shown in Fig. 5).

### 4. Generation of GUI elements

There are different techniques that enable defining position of GUI elements. Some Java layouts, HTML...
layers, XML layouts like XAML and other enable usage of absolute or relative coordinates to define positions. These positions could be specified to generator by attribute-value pairs, but that could be demanding, with too much coordinates to be specified.

The proposed solution in this paper uses iterators to define position of GUI elements. Iterators are post-processed after the main generation process is finished. Iterators have to be specified in both, Specification and Templates.

4.1 Specification of iterators in Templates

Iterators are specified in Templates within '#' signs, which is similar to connections, but uses the keyword iterator_, as shown in the following example:

```
<div style="position:absolute;top:#iterator_Y#%;
left:#iterator_X1#%; z-index:1">
    <b>#field_display#</b></div>
```

This example uses two iterators, #iterator_Y# and #iterator_X1#. On the other hand, the connection #field_display# is an ordinary connection that has its line in Configuration (here: #field_display#, field_display, means that #field_display# has to be replaced by the value of field_display, as specified in Specification). Iterators are directly connected to Specification, so there are no appropriate configuration lines.

4.2 Specification of iterators in Specification

Iterators have its separate area in Specification. Unlike other specification attributes, iterators are given by specifying two values, as shown in the following example:

```
iterator_Y:25,4
iterator_X1:35,0
iterator_X2:51,0
```

The first number right to iterator name is the starting value, and another is the increment value. Thus, the real value depends on the ordinary number of iterator occasion in the generated code, as shown in the example:

```
<div style="position:absolute;top:25%;
left:35%; z-index:1">
    Student id:</div>
```

```
<div style="position:absolute;top:29%;
left:35%; z-index:1">
    Surname and name:</div>
```

```
<div style="position:absolute;top:33%;
left:35%; z-index:1">
    Year of enrollment:</div>
```
As shown in the example, the value for \( Y \) coordinate starts from specified value of 25, with the increment value of 4. The \( X \) coordinate stays at 35. The generated user input/edit form is shown in Fig. 6:

![Fig. 6: Generated input/edit form](image)

### 5. XAML example

This example uses XAML to define the layout of Silverlight web application. The generated application is in C# and it is used for editing database table content by using input/edit form. The overall structure of the XAML document to be generated is given in its code template:

```xml
<UserControl
    xmlns="http://schemas.microsoft.com/.."
    xmlns:x="http://schemas.microsoft.com/.."
    xmlns:d="http://schemas.microsoft.com/.."
    Width="640" Height="480" mc:Ignorable="d">
    <Grid x:Name="LayoutRoot" Margin="0,0,8,0">
        #fields_entry#
    </Grid>
</UserControl>
```

The connection #fields_entry# is to be replaced during the process of generation with code defining edit fields and their labels. This fields could be different types, like integers, character strings etc, so corresponding configuration line defines usage of appropriate code template:

```xml
#fields_entry#,field_*,field_entry_*.template
```

Asterisk means “for all” (e.g. \( \text{fields}_* \) =all Specification attributes having name starting with \( \text{fields}_* \)). Specific suffix (e.g. number in \( \text{field_number} \)) is used to form filename of the template (here: \( \text{field_entry_number}.template \)). The template is defined for each type of edit field to be used in generation, e.g. \( \text{field_entry_number}.template \):

```xml
<TextBox x:Name="#polje#" Margin="286,#iterator_1#,234,0" VerticalAlignment="Top" TextWrapping="Wrap"/>
<dataInput:Label x:Name="#polje_labela#" HorizontalAlignment="Left" Margin="220,#iterator_2#,0,0" VerticalAlignment="Top" Width="62" Content="#polje_prikaz#:">
    #fields_entry#
    <Button x:Name="btnSave" Margin="312,230,268,0" VerticalAlignment="Top" Content="Save" Click="btnSave_Click">
        <Button.Background>
            <LinearGradientBrush EndPoint="6.519,4.273" StartPoint="6.308,4.318">
                <GradientStop Color="Black" Offset="0"/>
                <GradientStop Color="White" Offset="1"/>
            </LinearGradientBrush>
        </Button.Background>
    </Button>
</dataInput:Label>
```

As shown, two iterators are used in this template: \( \text{iterator}_1 \) to define \( Y \) coordinate of \( \text{TextBox} \) field and \( \text{iterator}_2 \) to define \( Y \) coordinate of appropriate label.

In Specification, these two iterators are specified as follows (numbers specify starting value and increment):

\[
\text{iterator}_1:38,28 \\
\text{iterator}_2:46,28
\]

Resulting layout of generated Silverlight application is given in Fig. 7.
Fig. 7: Layout of generated Silverlight application

6. Conclusion

This paper presents our solution for generating applications together with their GUI. For this purpose, our Dynamic frames generator model was improved for more effective specification of GUI layout. The solution is based on post-processing of generated source code by using iterators for specifying coordinates and other values of graphic elements. Iterators are tested on two examples. In the first example, GUI of web application is defined using HTML layers. Second example uses XAML to define the screen layout. For this purpose, a generator of Silverlight application was made.

It is shown that our solution enables effective specification of GUI layout, together with all other features of generated application.

7. References