WPFGen: A Code Generator for Rapid Development of Windows Presentation Foundation Applications

Aleksandar Bošnjak, Aleksandar Kaplar, Sebastijan Kaplar, Milorad Filipović, Željko Ivković, Sebastijan Stoja and Željko Vuković
University of Novi Sad, Faculty of Technical Sciences/Department of Computing and Control Engineering
Novi Sad, Serbia
{a.bosnjak.jr, aleksandar.kaplar, sebastijan.kaplar, mfili, zeljkoi, sebastijan.stoja, zeljkov}@uns.ac.rs

***Abstract*— This paper presents a language and a code generator called WPFGen for C#.NET WPF (Windows Presentation Foundation) applications. WPFGen enables iterative and incremental development providing an environment that supports rapid cycles of specification, code generation, and verification of invoked generated application. Manual changes in code are preserved.**

I. Introduction

Windows Presentation Foundation (WPF) [1] is a presentation framework for .NET applications that implements Model-View-ViewModel (MVVM) [2] design pattern (Figure 1). It provides a clear separation between application logic and the user interface (UI) appearance, which enables independent work of developers and designers. The appearance is specified in the Extensible Application Markup Language (XAML), an XML-based markup language, while the application logic is implemented in C#.NET or VB.NET classes.

An implemented WPF form consists of the following: (1) UI appearance specification written in the XAML file; (2) Code-behind – a class associated with XAML specification that implements the functionality that responds to user interactions; (3) a Model class, which provides representation of business domain entity; (4) a ViewModel class, which is the bridge between the view and the model [6].

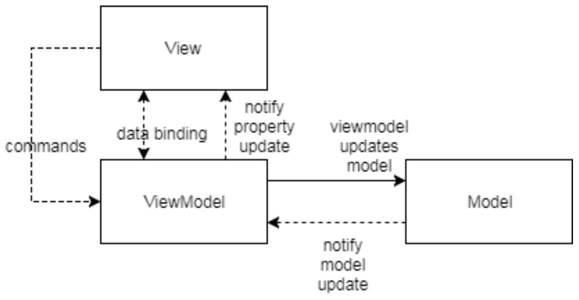


Figure 1: Model –View-ViewModel design pattern

Although powerful, WPF is complex, and the learning curve can be very steep. Creating a software product that meets the client's changing requirements in the short period of time can be a demanding job. This paper proposes a language that enables specification of forms at a higher level of abstraction, and a code generator called WPFGen for C#.NET. WPFGen generates WPF code based on specification created with clients. The generated code supports localization. Corresponding SQL scripts for database schema creation are also generated.

II. Related work

Before developing WPFGen, several existing solutions were analyzed based on both industrial environment and scientific community.

EasyCode [3] generates WPF forms based on database (DB) schema obtained via a data access layer (DAL). It provides create, read update and delete (CRUD) operations, search and document export features for all DB entities. DB schema is the only user provided input. The advantage of such approach is ease of use by the client who is generating a new WPF application. The disadvantage is that layout is not customable, and in case of many entity records, UI may seem opaque. EasyCode does not support localization.

DevExpress [4] uses the same client input like EasyCode and provides WPF forms with responsive layout. When an entity has many records, end users are able to reorganize their workspace, and manually set display for each WPF control. However, this only reflects to end users, and does not reflect code generation. Another important information regarding this solution, is that it is provided as an extension for Microsoft Visual Studio environment. Lead for such approach is ease of use for client. We thoughtfully considered using this approach but decided to build an independent application environment instead. The advantage we see in such approach is that our system won't be affected by changes in MS Visual Studio application programming interface (API).

Xomega [5] generates search or details views for WPF based on the type of their associated data object. This tool can only be used for generating view components for WPF applications - XAML.

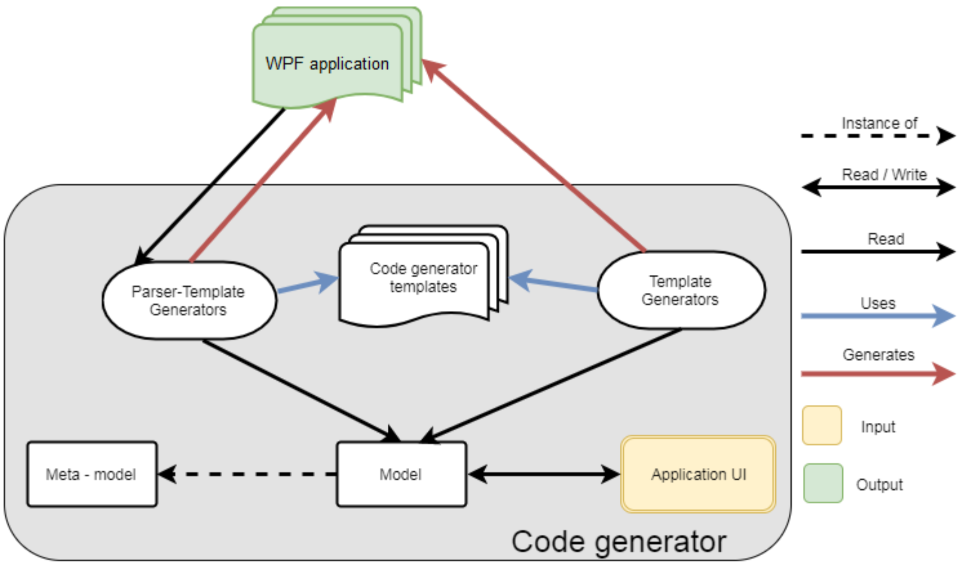


Figure 2: Main system architecture of WPF Gen

Xamarin XML Power Toys [6] goes a bit further from Xomega's solution [5] with ViewModel generation. It does not generate a model.

III. SYSTEM ARCHITECTURE

System architecture of WPF Gen can be described as association of the following set of components (Figure 2):

- Meta-model
- Model
- Application (WPF Gen) UI
- Templates
- Generators
- Output WPF application

The meta-model (Figure 3) is providing an abstract syntax for creating custom generated WPF applications. It allows *Product* creation by providing entity *Tables*, Columns on the one end, and custom *ViewLayout* for UI *Fields* on another. Besides that, meta-model allows grouping fields using *FieldGroup* feature. User is able to specify the output or input *File.*

Application UI (WPF Gen UI) is responsible for receiving input from the client in order to create the specification (concrete model state). Even though textual description of a model is often more expressive, we have built separate application UI for creating the model so that our application is easier to use for the client.

The model is an instance of meta-model. It presents concrete meta-model state for ongoing generating *Project*. Based on model state, generators are providing relevant output.

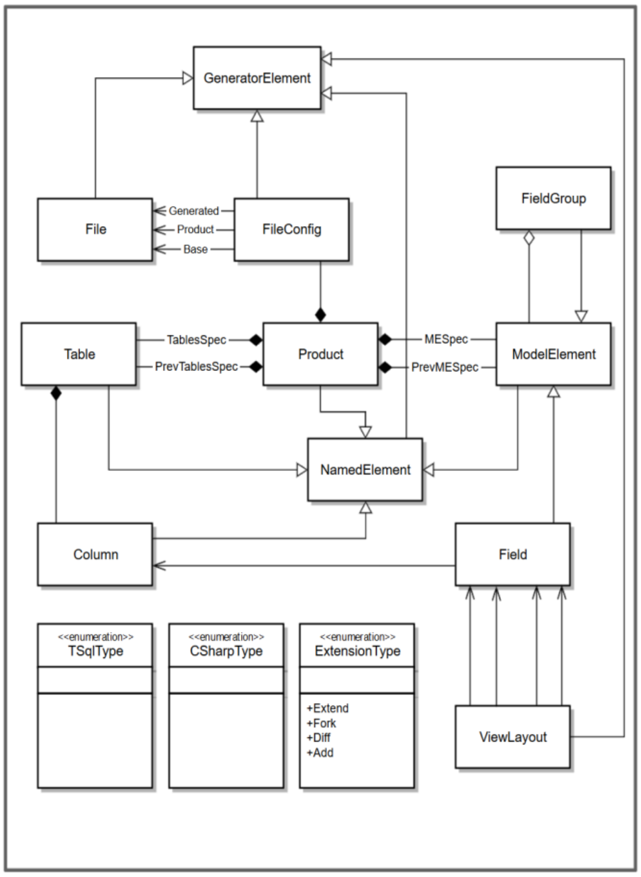
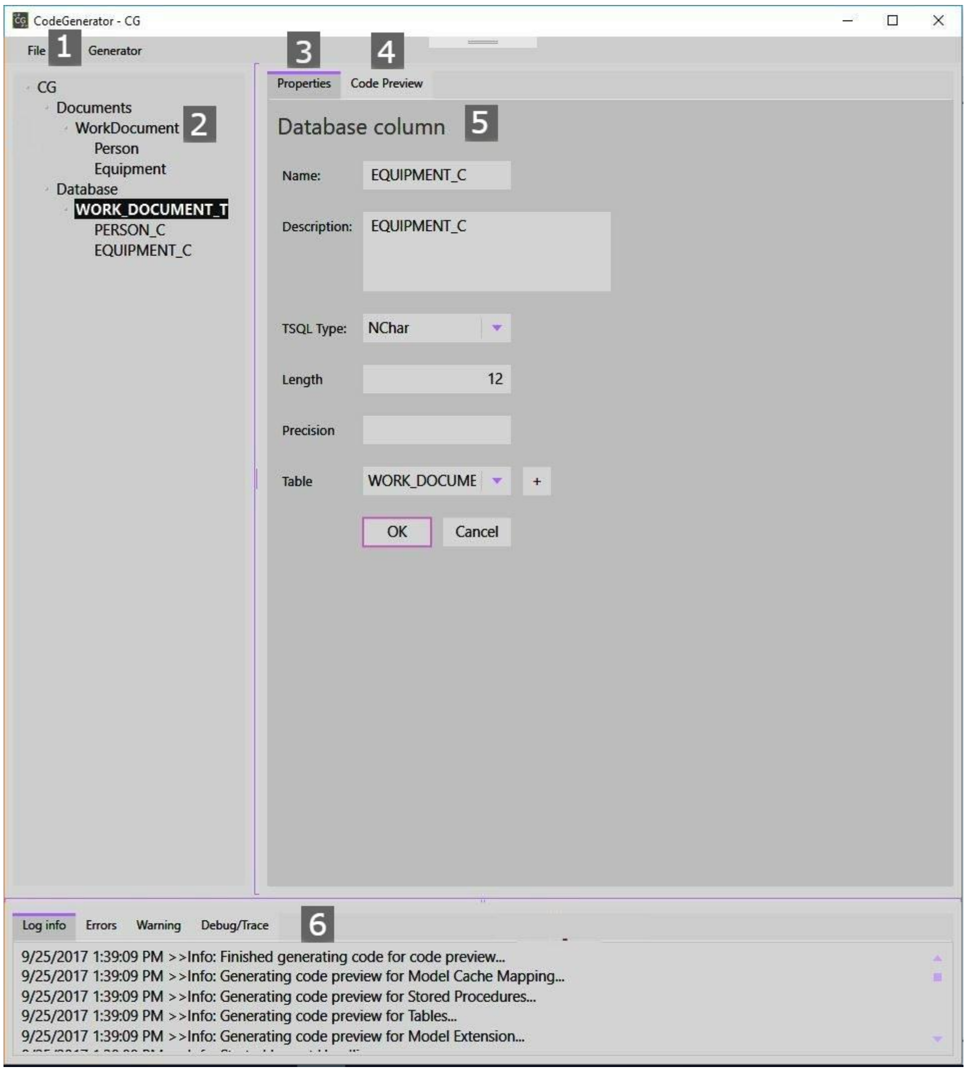


Figure 3:A high level view of a meta-model of the language for WPF forms specification



Name: EQUIPMENT_C

Description: EQUIPMENT_C

TSQL Type: NChar

Length 12

Precision

Table WORK_DOCUME

```
9/25/2017 1:39:09 PM >>Info: Finished generating code for code preview...
9/25/2017 1:39:09 PM >>Info: Generating code preview for Model Cache Mapping...
9/25/2017 1:39:09 PM >>Info: Generating code preview for Stored Procedures...
9/25/2017 1:39:09 PM >>Info: Generating code preview for Tables...
9/25/2017 1:39:09 PM >>Info: Generating code preview for Model Extension...
```

Figure 4:WPFGen development environment. 1. Main menu, 2. Forms selection tree, 3. Selected form property panel, 4. Code generation section, 5. Selection panel, 6. Logging panels.

Once a model has been created, the *Parser-Template generators* and *Template generators* are able to generate relevant C# code using *Code generator templates.* Template generators generate C# code for Models and ViewModels using T4 template engine [7]. Generated C# code is in partial classes, providing support for manual changes in the other parts of the classes, without the need to develop a dedicated mechanism for the preservation of the handwritten code. Parser-Template generators also uses T4 template engine for generation of parts of View (XAML files), local resources (.resx file), and project (.csproj file) entries, but generated code is inserted in proper places using Extended XML parser [8], taking care of manual changes, comments, and file formatting. The result of WPF Gen is a complete Visual Studio C# project, which can be built and executed.

IV. CODE GENERATOR

To enable collaborative development, WPFGen provides a development environment that supports rapid cycles of the specification, code generation, and verification of invoked generated application. This includes implies:

- a method of specification familiar to the end-users (through WPF forms)
- validation of specification with reporting warnings and errors
- code preview
- generation of code that can be invoked immediately
- iterative and incremental development, with preserved manual customization of generated code.

WPF Gen development environment (Figure 4) consists of the following:

1. Main menu
2. Selection tree
3. Selected form property panel
4. Code preview/generation section
5. Selection panel
6. Tracing panel for logging generation process, errors, warnings, etc.

The selection tree provides an overview of the project. Using the right mouse click, the client is able to add or remove entities.

Selected property panel is used for adding new and modifying existing selected nodes.

Code preview/generation section allows client to see a preview of the code which will be generated. It shows a list of files, their paths and generated source code of the selected file. Having an insight into this allows client to copy and partially use generated code in some cases. If code has previously been generated, the code preview panel also shows a diff view between the previously generated code and code which will be generated with the current model.

Tracing panels are informative. They provide all necessary information about:

- Info – generation process itself, containing each generator action measured in time.
- Error – information about errors that have occurred during the generation process.
- Warning – information about possible warnings about code generation
- Trace – used for developing purposes and only visible if WPF Gen is built in DEBUG mode.

V. Case study

In order to provide concrete results, we have provided development time comparation between eight-person team of developers and WPF Gen. In order to do so, we explored two scenarios:

1. Generating from scratch (no earlier development is being done regarding generation scenario, Table 1)
2. Update/extend already generated/developed files (Table 2)

	eng/days for manually	eng/days with code generator	Number of files	Number lines code
Model + configuration	2	0.5	5	1570
ViewModel	15	1	8	970
Xaml	15	1	6	820

Table 1: Test case for customization of one field for one document from scratch

	eng/days for manually	eng/days with code generator	Number of files	Number lines code
Model + configuration	2	0.5	5	1570
ViewModel	15	1	8	970
Xaml	15	1	6	820

Table 2: Case study for customization of one field for one document update

VI. Conclusion

This paper presents a language and a code generator called WPFGen for C#.NET WPF (Windows Presentation Foundation) applications. It provides iterative and incremental development including an environment that

supports rapid cycles of specification, code generation, and verification of invoked generated application.

WPF Gen allows the user to specify entities and their field through the provided user interface. Using this model, WPF Gen generates Models, ViewModels and XAML files, along with a C# project file. The resulting source code can be built and executed. After seeing the results, the user can change the model and repeat the code generation process. Manual changes of the source code by the user are also supported and will not be lost when code is generated again.

Compared to existing solutions, WPF Gen allows the user to make further edits to their model, provides the possibility of reorganizing visual appearance of UI components in the resulting application, supports localization and generates a solution that is closer to being complete.

Further research would include support for validation of generated forms. This would increase the robustness and quality of the generated application.

VII. References

[1] Microsoft, "Introduction to WPF," [Online]. Available: https://msdn.microsoft.com/en-us/library/aa970268(v=vs.100).aspx.

[2] Microsoft, "The MVVM Pattern," [Online]. Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx.

[3] EasyCode, "http://www.eazycode.com/index.htm," [Online].

[4] DevExpress, "https://documentation.devexpress.com/WPF/115192/Scaffolding-Wizard/UI-Generation," [Online].

[5] Xomega, "http://www.xomega.net/Generators/UICS/WPFViews.aspx," [Online].

[6] Xamarin, [Online]. Available:

https://blog.xamarin.com/power-visual-studio-using-xaml-power-toys.

[7] Microsoft, "T4 template text templates," [Online]. Available: https://msdn.microsoft.com/en-us/library/bb126445.aspx.

[8] A. Lukic, "Extended XML Parser," [Online]. Available: https://github.com/lukic-aleksandar/XmlParser.XPath.