MCDA4ArcMap Architecture and Implementation Details

Table of Contents

Setup 3

Configuration File 3

Event handling 4

Tool Parameter 5

Transformation Strategies 6

(MCDA)Tools 7

MVVM 8

Rendering 9

Miscellaneous 10

Conclusion 11

List of Figures

Figure 1. The implementation *ToolParamer* offers *Properties* and functions to describe criterions 6

Figure 2. The relation of the *TransformationStrategyFactory* with the *ITransformationStrategy* interface and the two implementations. 7

Figure 3. The *ToolFactory* offers three methods to create new tools with an initialized data table. 7

Figure 4. Every tool inheriates from the *AbstractToolTemplate* class. An example of the template pattern. By declaring abstract methods in the *AbstractToolTemplate* class all subclasses have to implement these methods. Finally the whole algorithm works by calling the *Run* method which calls than the abstract methods – or algorithm fragments. 8

Figure 5. The Model View ViewModel (MVVM) concept with the *WLCTool* as an example. 9

Figure 6. The relationship between all render components. 10

## Setup of Development Environment

The *add-in* was written in *C#* and built upon the *ArcObjects SDK 10* for *.NET* and the *.NET Framework 4.0.* The choice of the *.NET Framework* version and the version of *Microsoft’s Visual Studio* depends on the support from *ESRI*. The current target *ArcMap* version is *ArcMap 10.1.*

*ESRI* offers an up to date compatibility matrix[[1]](#footnote-1) for their *SDK* and *Microsoft* products. It is noteworthy that it is possible to develop *add-ins* with the *Microsoft Visual Studio Express Edition*, which is basically free for non-commercial products.

The project is hosted on *codeplex[[2]](#footnote-2)*. As the used framework, the integrated development environment and the hosting platform are all *Microsoft* based. Therefore, it makes sense to use the *codeplex* *Team Foundation Server* as version control system.

First of all one should install the latest supported version of *Microsoft Visual Studio.* Once completed, the *ArcObjects SDK 10* for *.NET* needs to be installed. In the *Source Code* section of *codeplex* exists a *Connect* button which offers the link to a tutorial and the required server name and user credentials to checkout the code.

The *add-in* has one external dependency. The *Extended WPF Toolkit[[3]](#footnote-3)* library is required to build the project. One way to manage dependencies is the *Microsoft Visual Studio* plugin for *nuget*[[4]](#footnote-4), a package manager for the *.NET Framework*.

## Configuration File

The graphical user interface components, their structure as well as some additional information can be defined in the configuration file. The configuration file was introduced by *ESRI* especially for the development of *add-ins*. It has the file extension *esriaddinx* and is *XML* based.

The first lines are for general information like the version or the author name of the *add-in*. This information can be found during the installation and in the extension menu in *ArcMap*.

<Name>MCDA Add-in</Name>

<AddInID>{f098ac0f-6cae-4e1b-81d0-d67452d3cf57}</AddInID>

<Description>Add-in for MCDA analysis</Description>

<Version>1.0</Version>

<Image>Images\MCDA Add-in.png</Image>

<Author>Steffan Voss</Author>

Components for the graphical user interface which were defined in the configuration file, are accessible in other classes like normal code.

UID dockWinID = new UIDClass();

dockWinID.Value = ThisAddIn.IDs.AddDataView;

IDockableWindow w = ArcMap.DockableWindowManager.GetDockableWindow(dockWinID);

w.Show(true);

For example the *AddDataView* class *ID* can be accessed via the *ThisAddIn* class in order to show the window. This is possible because the components defined in the configuration files were compiled into code.

Windows defined in the configuration file follow the *Singleton Pattern*. Therefore, it is impossible to create more than one instance at a time. For *MCDA4ArcMap* it is necessary to have multiple tool windows. One solution is the creation of windows with the *WindowInteropHelper*. As a result some windows are defined in the configuration file and others in the corresponding button.

The disadvantage of windows that are not defined in the configuration file is that they cannot be pinned in *ArcMap*. Moreover, these windows cannot be blocked from *ESRIs IProgressDialog2,* which is used for long running tasks.

Basically everything that is defined in the configuration file can be defined with the help of wizards.

## Event handling

The *Model*, especially the class *MCDAExtension,* notifies *ViewModel* classes and others about new events like removed or added feature classes. To improve the granularity others can register for a single *Property.* This is useful for performing updates only for specified changes and therefore, to improve the performance. The proposed way to handle such events in *C#* misses a few crucial features. The approach to notify is actually based on calling a method with the name of the *Property* as a string*.* This technique leads to errors, as the compiler cannot verify if the used string of an intended *Property* even exists. In order to improve this approach a method extension exists. This extension expects the actual *Property* as an argument. Typically one member variable is set at the same time.

An example is the *AvailableLayer Property* of the *MCDAExtension* class. The *Property* is supposed to notify all registered listeners in case someone changes the value of the *Property*.

public IList<Model.Layer> AvailableLayer

{

get { return \_listOfAvailableLayer.OrderBy(f => f.LayerName).ToList(); }

set { PropertyChanged.ChangeAndNotify(ref \_listOfAvailableLayer, value, () => AvailableLayer); }

}

The *ChangeAndNotify* method expects

* the member variable
* the value
* the *Property* to notify

However, to use the *ChangeAndNotify* method the class should implement the *INotifyPropertyChanged* interface for the *PropertyChangedEventHandler* event. In the example the name of the object is *PropertyChanged.*

Everyone who wants to listen to such an event can register to a specific *Property.*

*\_*mcdaExtension.RegisterPropertyHandler(x => x.AvailableLayer, MCDAExtensionPropertyChanged);

The *RegisterPropertyHandler* method expects

* the *Property* (checked by the compiler)
* the method to call in case the *Property* changed

The variable *\_mcdaExtentionsion* is an instance of a class that implements the *INotifyPropertyChanged* interface. The *Property AvailableLayer* notifies everyone who listens in case the setter of the *Property* is called.

## Tool Parameter

The *IToolParameter* interface and the implementation *ToolParameter* represent the parameter and information collection for every criterion. A class diagram can be found in Figure 1.

The *Property* *LastWeightChangedToolParameter* is a reference to the *IToolParameter* which was last changed by the user or the system. This reference is important for the *ProportionalDistributionStrategy*,the only implementation of the *IWeightDistributionStrategy*.The purpose of this strategy is to ensure the following condition: where is a *IToolParameter* weight. When implementing this condition the last changed parameter is required to exclude it from the adjustment if possible.

A potential pitfall is the fact that the *LastWeightChangedToolParameter Property* is static and the *ToolParameter* implementation is used for all instances created by the *ToolFactory*. Hence, if the user or the system changes two or more *IToolParamter* at the same time the algorithm may use the wrong *LastWeightChangedToolParameter*. In the current implementation this can be avoided, as the user cannot use two or more *ITool* implementations parallel, since the user has only one input device.

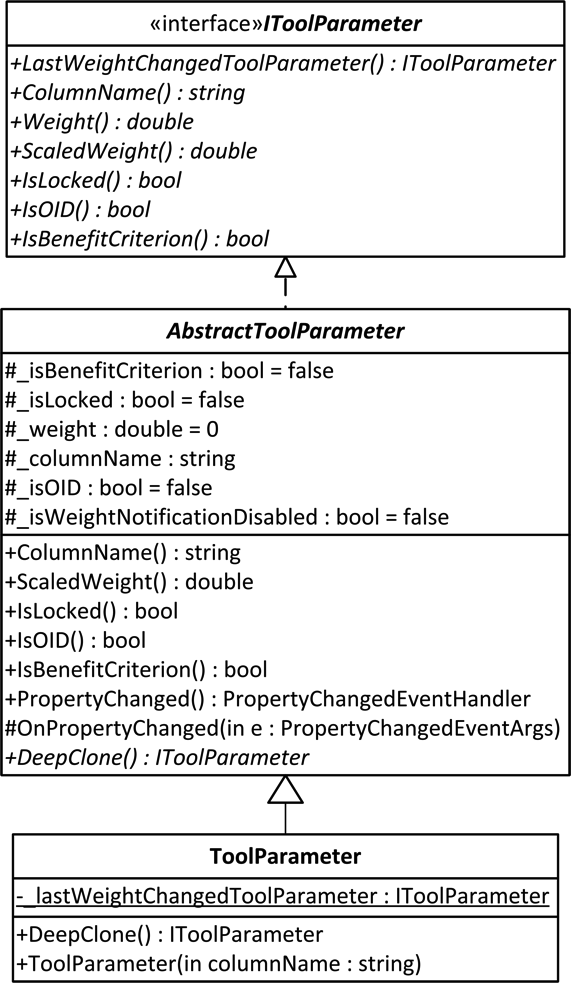


Figure 1. The implementation *ToolParamer* offers *Properties* and functions to describe criterions

The *DeepClone* method is important for the *RenderOption AnimationLike*. Once the user finished the analysis the system has to select the states required for a smooth animation. In order to do so every state needs to be saved. Calling the *DeepClone* method for every state of every *IToolParameter* does this.

## Transformation Strategies

In order to apply *MCDA* methods to different criteria variables it is necessary to norm the values. Otherwise the outcome would depend on the range of the values and not on the weight. As this part is essential the template for any *MCDA* algorithm has a method called *PerformScaling,* which has to be implemented by every algorithm. However, in case this step is not required the programmer can simply leave the method empty.

There are several strategies to perform this transformation. An overview of the involved classes, interfaces and enumerations can be found in Figure 2. To make these strategies interchangeable the *ITransformationStrategy* interface was introduced. The tool offers with the *MaximumScoreTransformationStrategy* and the *ScoreRangeTransformationStrategy* two popular implementations of transformation strategies. Both can be accessed via the *TransformationStrategyFactory.* In practice the abstract tool template comes with a *Property* for the *TransformationStrategy Enumeration*. Thus, every tool is implemented in a way that allows the user to change the *TransformationStrategy* at runtime.

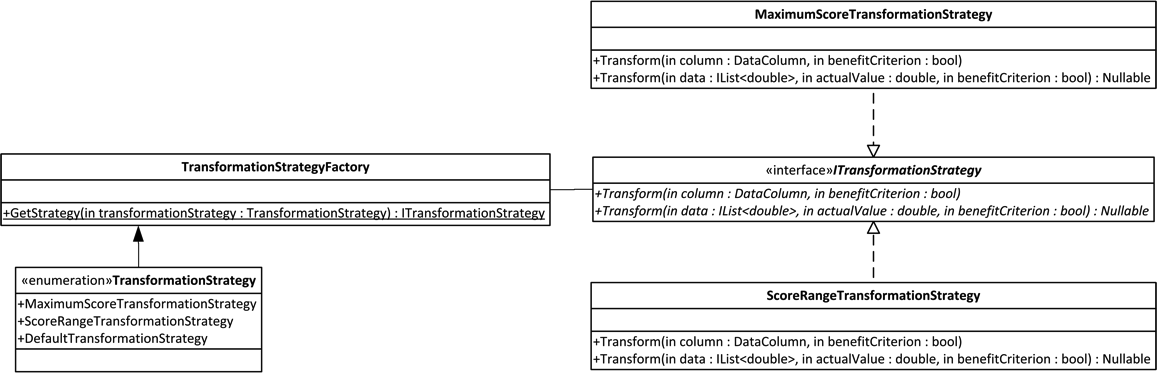


Figure 2. The relation of the *TransformationStrategyFactory* with the *ITransformationStrategy* interface and the two implementations.

The *ITransformationStrategy* interface itself specifies two methods. The intention of the method with the *DataColumn* parameter is to transform a complete column based on the maximum or minimum value inside the column. The other option let the user pass a list of values and a value which works as the value of interest. In contrast to the *DataColumn* parameter method the result is a single transformed value. Thus, the method is useful whenever it comes to self-defined cluster of values.

## (MCDA)Tools

A tool in the context of the *MCDA4ArcMap* project is a *MCDA* method. *Tool* objects can be created with the help of the *ToolFactory.* An overview can be found in Figure 3. One benefit of the *ToolFactory* is that all tools are initialized with the current data and the selected criterions.

The structure follows the template pattern. By following this pattern an abstract class defines methods that are part of the (*MCDA*) algorithm. It also implements a method – in this case the *Run* method – which calls all abstract methods. As a result the *Run* method call is equivalent to an algorithm run. This approach is visualized in Figure 4.

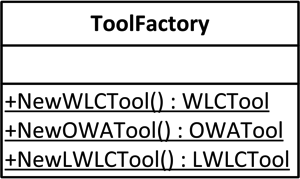


Figure 3. The *ToolFactory* offers three methods to create new tools with an initialized data table.

Besides the methods to define the algorithm fragments some additional *Properties* are declared as abstract. Mandatory are *Properties* for the *TransformationStrategy,* the *ToolParameterContainer* and the *DataTable.* The purpose of the *TransformationStrategy Property* is the interchangeability of transformation strategies at runtime. This is the case whenever the user changes the strategy in the view. By calling the *Run* method afterwards up to date results can be guaranteed. With the *ToolParameterContainer* and the *DataTable* the internal data can be accessed. Both *Properties* are set when creating a new tool by the *ToolFactory*. As it is only possible to extract the tools to a certain extent every tool has it specific *Properties* and methods.

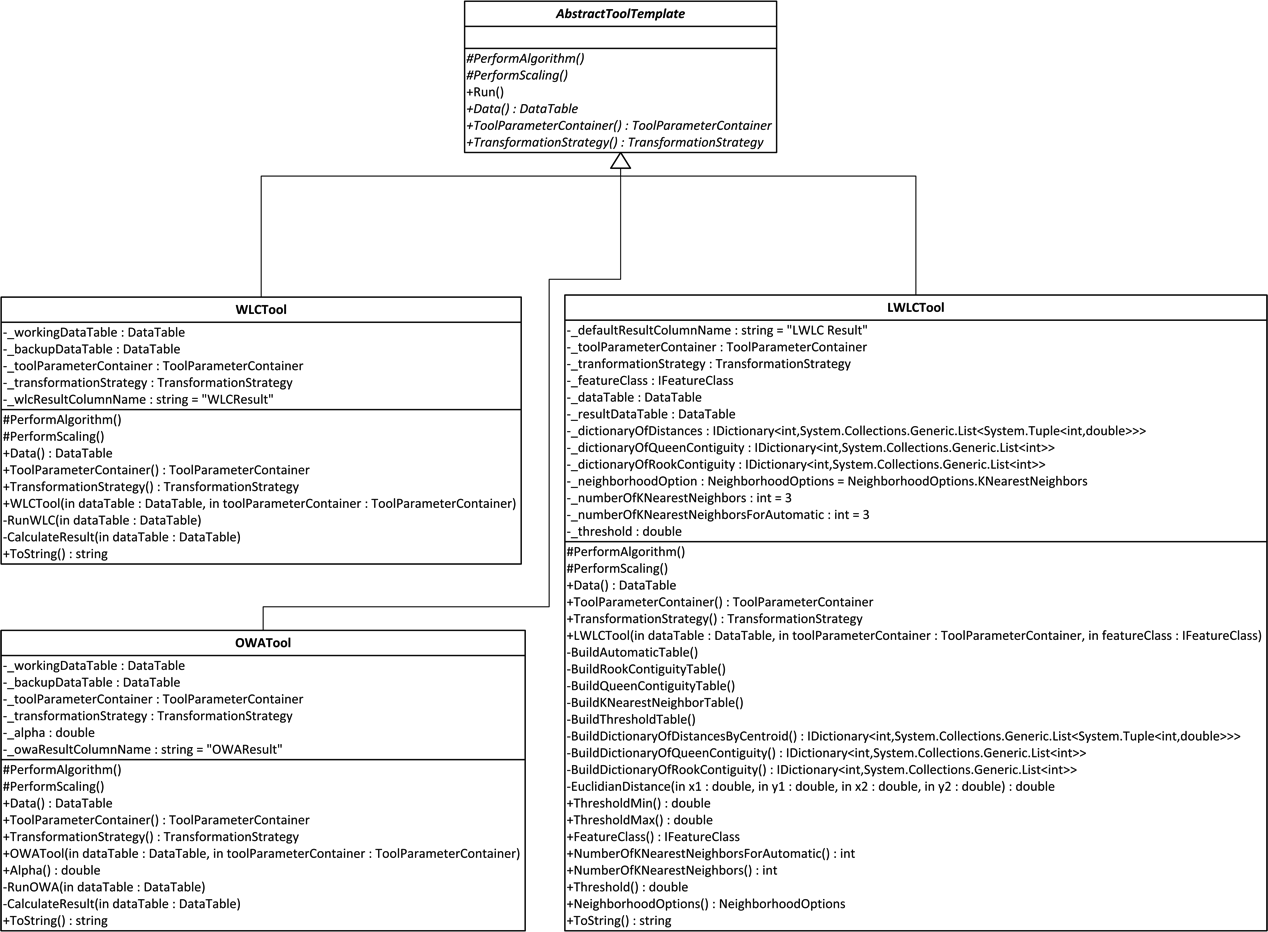


Figure 4. Every tool inheriates from the *AbstractToolTemplate* class. An example of the template pattern. By declaring abstract methods in the *AbstractToolTemplate* class all subclasses have to implement these methods. Finally the whole algorithm works by calling the *Run* method which calls than the abstract methods – or algorithm fragments.

All *Tools* are optimized for multi-core CPUs. The sophisticated *C# Parallel library* makes it possible to run simple *for* loops or even complete methods parallel – with a single method call. Besides the easy to use *Parallel* library it is still the responsibility of the programmer to take care of locking critical sections. A mature problem is that the insert functions of the underlying data table are not thread safe. As a consequence, it is possible to run all calculations parallel, but the merge operation is still a bottleneck. Despite the mentioned bottleneck the distribution of the calculations speeds up the program tremendous.

## MVVM

The proposed pattern for graphical user interfaces based on *Microsoft’s WPF (Windows Presentation Foundation)* is the *Model View ViewModel* pattern. In principle the *add-in* follows this pattern whenever it is appropriate. The actual components for the user interface are written in *XAML (Extensible Application Markup Language)*. Together with the *Code Behind Page* they define the *View*. An example from the *WLCTool* can be found in Figure 5. Except for a few very specific methods only generated code exists in the *Code Behind Page*. The connection between the *View* and the *ViewModel* is established via *Data Binding.* In contrast to the *Data Binding* the action which is performed by pressing a button is encapsulated into an instance of *ICommand*.

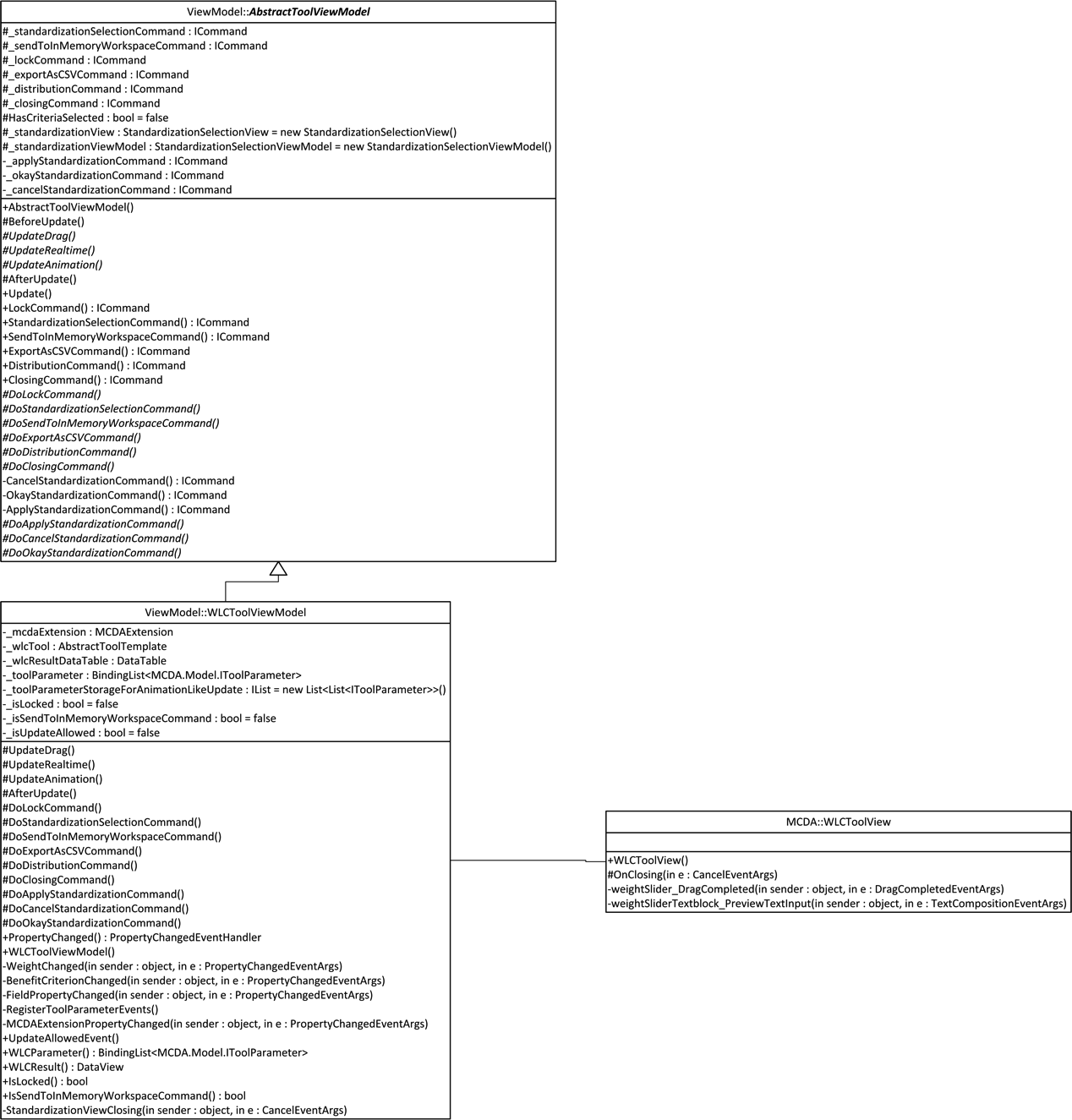


Figure 5. The Model View ViewModel (MVVM) concept with the *WLCTool* as an example.

A few *ICommands* are the same for every *Tool*, like the export *ICommand.* These *ICommands* and others are listed in the *AbstractToolViewModel* class. By inheriting this class all *ViewModels* share this common code base.

## Rendering

As the *MCDA4ArcMap* project is highly dynamic the rendering is an important part. *ESRI* provides numerous renderer classes. Basically the provided renderers are used and only the input argument and update mechanism is customized. The user has the choice between the *ClassBreaksRenderer*, the so-called *BiPolarRenderer* and the default *SimpleRenderer*.

In order to achieve the desired dynamic behavior the *MCDA4ArcMap* project has to perform an update on the rendering whenever the user changes the input. Therefore, the *Render* method is called when a *Tools* *Run* method is called.

Several classes are part of the rendering mechanism. An overview can be found in Figure 6. The class in the center is the *RenderContainer*. One *Property* of the *RenderContainer* is the *Renderer* enumeration. As the *RenderContainer* can host the data for the *ClassBreaksRenderer* and the *BiPolarRenderer* it is essential to define which one has to be rendered. The *None* member of the *Renderer* enumeration is equivalent to the *SimpleRenderer*.

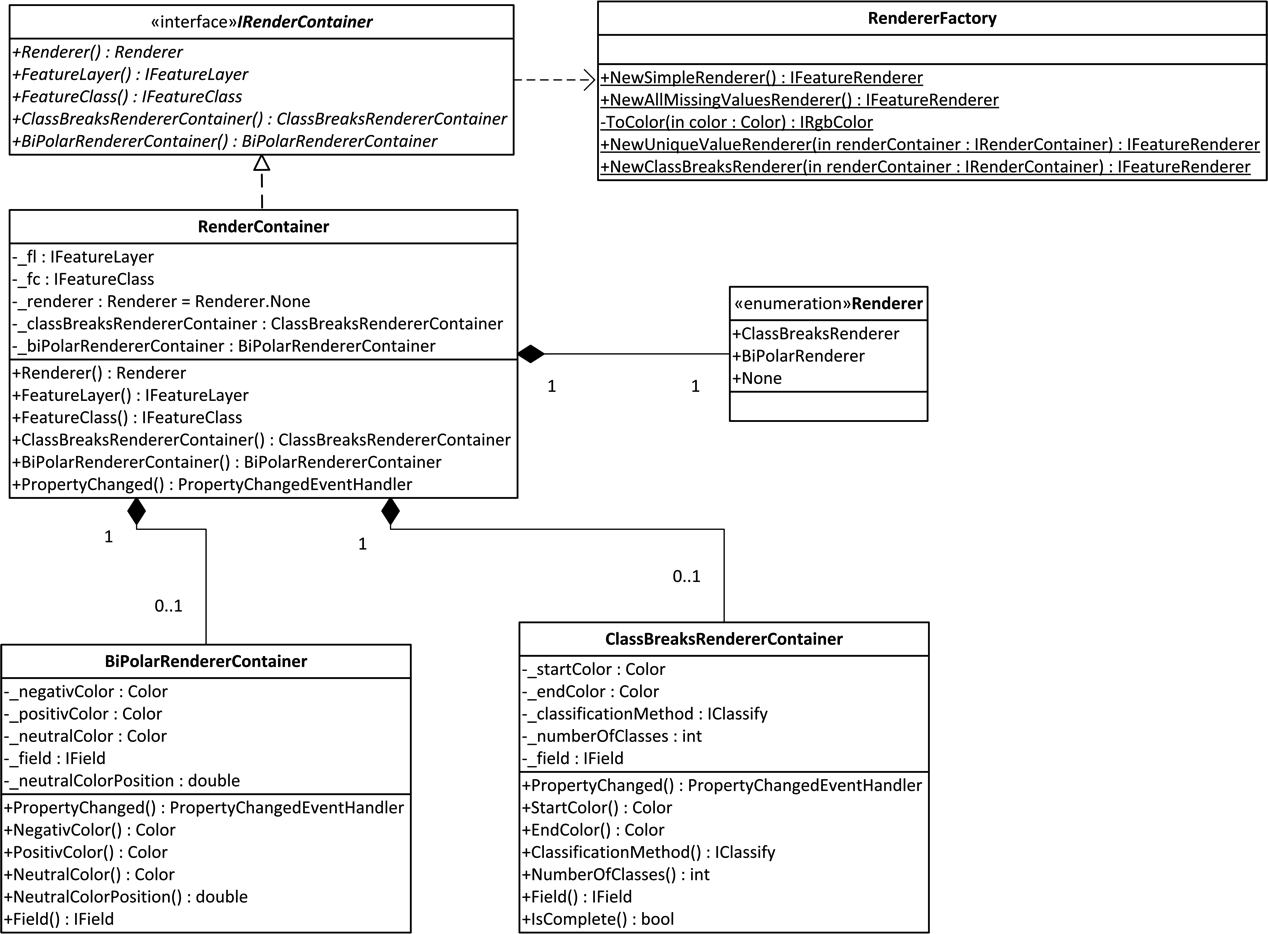


Figure 6. The relationship between all render components.

The required data for the *ClassBreaksRenderer* and the *BiPolarRenderer* differs in almost all aspects. One solution was the creation of two container classes for each renderer. The link between *ESRIs* renderer and the container is made via the *RenderFactory*. The *RenderFactory* expects for the more complex renderer an argument of the type *IRenderContainer.*

After each *Run* call of a tool the *RenderFactory* is used to create a new renderer based on the selected rendererfrom the *Renderer* enumeration. Subsequently, the recently created renderer is set as the new renderer for the feature layer and a partial refresh is performed.

All arguments for the renderer can be set in the *Visualization* view, a part of the graphical user interface. The render containers are also used to restore the specific render settings when switching between the available layers in the *Visualization* view.

## Miscellaneous

The external dependency *Extended WPF Toolkit* is required to offer the user a color picker component in the visualization window. *ESRI* provides a component for selecting a color, color ramps or both. However, these controls are developed for the *Windows Forms* framework and do not work in combination with *WPF (Windows Presentation Foundation)* windows. In theory it is possible to host such legacy *Windows Forms* controls in *WPF* applications.

For an unknown reason this does not work with the *ESRI* components. Other developers also reported this problem. Another difficulty is the fact that the external dependency has to be referenced inside a code file. Usually it would work to define the color picker only in the *xaml* file of the *Visualization* window and to bind it in the corresponding *ViewModel*.

Nevertheless, the application crashes because the external dependency cannot be found. This approach works with all other kinds of applications, but not for *add-ins*. A workaround is to define the color picker in the *code behind page* and manually add the control to a layout component defined in the *XAML* file. As one consequence it is necessary to define the binding in the *XAML* file too. After performing these steps it is possible to define the binding for the color picker in the *ViewModel* as usual.

In order to offer the user ways to analyze the data on his own it is possible to export the complete data table that is used for calculations and the corresponding arguments into a *CSV* file. It is notable that the export methods in the *Export* class take the actual *Property* name with the help of the reflection capabilities as string for the output.

For the sake of the user data safety and integrity all *Tools* work on in-memory copies of the actual *shapefile*. That the original data is not touched under any circumstances is based on the fact that the locking mechanism for tables is too slow in an *ArcObjects* environment. It can cost up to several seconds to lock and unlock a table. In contrast the update of the result column has to be done almost in real-time.

## Conclusion

For large parts the code should be self-explaining. In case the situation is complicated or a workaround was implemented an explaining comment was added in the code. *ESRIs ArcObjects* library has many design flaws. This may be the results of adding new functionality over the years and supporting legacy products. Sometimes the fastest way to add new features was the trial and error approach.

In case there are questions please feel free to contact me: *steffan.voss@uni-muenster.de*

1. http://resources.arcgis.com/content/arcgissdks/10.0/system-requirements [↑](#footnote-ref-1)
2. http://mcda4arcmap.codeplex.com/ [↑](#footnote-ref-2)
3. http://wpftoolkit.codeplex.com/ [↑](#footnote-ref-3)
4. http://nuget.org/ [↑](#footnote-ref-4)