Approaches to designing the graphic user interface components for mobile SCADA HMIs and application

BADEA FLORIN, POP EMIL
University of Petrosani
Str. Universitatii nr. 20, Petrosani, HD
ROMANIA
flo_badea@yahoo.com, emilpop2001@yahoo.com

Abstract - This paper presents approaches to designing a user interface (UI) component framework that can be used to speed up the development of mobile SCADA HMIs. The paper presents the supported features and the design consideration behind each of the components contained in the framework. The framework has 3 versions that were built to support 3 of the most popular mobile platforms on the market today (Android, WP7 and Blackberry). After the design considerations, the paper discusses each framework’s flexibility as well as the extensibility possibilities. At the end we present a client application that uses the framework in order to monitor the heat distribution system installed in a university campus.

Keywords: SCADA, Mobile Device, UI Components, WP7, Blackberry, Android

1 Introduction

With the rapid expansion of smart phones in the last couple of years, application development demands for these mobile platforms also increased. Nowadays application requirements are more and more complex and there is an increasing need to speed up the application development process in order to comply with the tight deadlines and the rapid change in development technologies. This is also true when it comes to the development of user interfaces for SCADA applications.

With more and more remote data acquisition functions moving into the cloud, now is the best time possible to explore what mobile SCADA can do for the industry. Wireless SCADA has taken off in recent years, and is in use especially in industries such as the solar and wind power industries, where devices are often running in difficult to reach, rural settings.

SCADA HMIs have only recently begun to emerge on mobile devices. The very first mobile SCADA applications were either simple text based or web based (written with HTML and JavaScript) clients that lacked advanced UI components. New development technologies and frameworks like the Android SDK, Silverlight, RIM SDK and iOS SDK have made it possible to develop rich client applications for mobile devices that present amazing user experiences. These technologies and frameworks can now be used to build rich mobile SCADA HMIs.

At the moment there are a number of solutions available for data visualization on mobile devices. There are some general solutions (not targeted specifically for SCADA applications) and there are some SCADA specific ones as well.

General data visualization solutions for mobile devices are available from Telerik, ComponentArt and ComponentOne among others. These solutions are Silverlight based and are designed to run on WP7 devices. These frameworks contain charting controls, gauges, maps, data grids, time navigators and dashboard controls. CSWorks offers a UI framework that is specifically targeted for SCADA HMI design. This framework is also Silverlight based and includes UI controls for pipes, tanks, valves and pumps.

This paper will present approaches to designing a number of UI frameworks that can help speed up the development of mobile SCADA interfaces for some of the most popular mobile platforms available on the market today (Android, WP7 and Blackberry).

2 The UI component framework

Each version of the framework will contain roughly the same set of UI components. These are components that are widely used in every SCADA HMI. The list of components offered by the framework includes: gauges, tanks, pipes, valves and pumps. The following sections will describe in detail the design considerations behind each component.
2.1 The Gauge Component

The gauge component should present a scale and permit multiple indicators to point to particular values within that scale [5,6]. A scale should contain ticks (minor and major), labels, and ranges. The ranges are used to assign different colors to different intervals in the scale. These ranges can be used to indicate below optimal, optimal and above optimal values.

There are different types of gauges that can be implemented. The frameworks presented here implement linear and radial gauges. When thinking about gauges, some common features come to mind that should be exposed by each type of gauge. Some of these features relate to the behavior of the gauge while others relate to the gauges’ appearance. The gauge class hierarchy can be seen in figure 1.

![Fig.1 the Gauge class hierarchy](image)

As can be seen from the above image, the tick placement will be specified in the derived classes. This is in order to further clarify the meaning of the properties. A single property in the base class with more options in the enumeration would have confused the user [5,6].

The Scale indicators will have at a minimum 2 properties: the Value property and the Owner property. This last property (Owner) identifies the scale that owns the indicator. Other properties might be needed depending on the framework version. Silverlight, like WPF, promotes a new concept in application development. This concept separates control behavior from its appearance. In these frameworks the controls are lookless, meaning that the developer can change the appearance of the control without needing to modify its behavior. This is achieved by means of control templates [3,4]. The WP7 gauge implementation takes advantage of this and many other Silverlight features in order to present very flexible controls. Figure 2 shows the WP7 gauge controls. This is another reason why the WP7 versions don’t need too many properties that customize the control appearance. These can be specified in the control template.

![Fig.2 WP7 gauges](image)

In the Blackberry and Android versions, the gauge controls take care of all the drawing. They draw the scales, ticks, ranges and the indicators. This is totally different from the WP7 version, where the gauge was a collection of smaller components that drew themselves. This implementation limits the flexibility of the scales. If there is a need to change the layout of the gauge even a little, the developer needs to override the drawing code in order to change the UI [1,2]. Figure 3 shows some samples of the Blackberry and Android gauge components.

2.2 The Tank Component

The storage tank is a graphical element that shows a fill level. The tank is a lot like a linear scale.
in that it presents how much of it is filled. The tanks contained in the frameworks offer the following basic properties: **Capacity** and the **CurrentLevel**. Besides these mandatory properties a tank may also expose properties used to enhance its visual appearance. These properties are dependent on the framework for which they are implemented.

For the WP7 implementation, the tank will have only the 2 properties listed above. The rest of the features will be implemented in the control template. Figure 4(a) presents a few customized WP7 tank components.

![Fig.3 BB and Android gauges](image)

For the Blackberry and Android versions we need to add more properties to the Tank class. These properties will be used to customize the appearance of the tank and are added to compensate for the apparent lack of flexibility in these frameworks. We added properties to show the level indicator, the label and the mask. We also added properties for the position and size of the level indicator and for the tank color among others.

A lot of properties have been added to the tank class just to support a few UI features. If we wanted to add new UI elements or maybe change the shape of the tank, we would have to derive from the Tank class to add them. Then we would need to override the drawing code which would require intimate knowledge of the control structure. Figure 4(b) shows some samples for the Blackberry (left) and Android (right) Tank components.

### 2.3 The Pipe Component

The pipes are probably the simplest controls implemented in the frameworks. The pipe control has only 3 states: full, empty and indeterminate. Based on the current state, the pipe will change its color. In order to implement the pipes a class hierarchy was built. The hierarchy was built mainly to support the different pipe shapes. The pipe hierarchy structure depends on the framework for which the pipes were built. Figure 5 shows this hierarchy.

![Fig. 5 pipe component class hierarchy](image)

For the WP7 pipes we need to add more classes to the hierarchy. This is because we have 4 elbow pipes and 4 T pipes. We will add these new classes without any additional features. We will use them only to apply different control templates depending on the type of pipe we want to obtain.

For the Blackberry version of the pipes, the **ElbowPipe** and **TPipe** classes take care of drawing the different pipe orientations based on the **Orientation** property. The Android pipe design is almost identical to the Blackberry one from the appearance point of view. The differences are in the implementation. Since the Blackberry API does not support a rotate transform, each elbow and T pipe was calculated manually in the drawing code. In the Android implementation only one type of elbow and one type of T pipe were calculated. The rest were obtained by rotating the canvas before drawing. Figure 6 shows the pipe components.

![Fig. 6 pipe components](image)
2.4 The Pump Component

A single type of pump has been built, a centrifugal pump. The WP7 the pump blades are rotated using an animation. The animation is started when the pump is in the Running state. When the state of the pump changes to Running the visual state manager starts a storyboard that animates the Angle property of a RotateTransform. This RotateTransform object sets the value of the pump blades' RenderTransform property.

For the Blackberry version, the blades are rotated by using a timer with a predefined interval. Since the API does not support transforms, the points on the blade need to be calculated every time we need to rotate the blade.

For the Android version, the blade points are calculated only once and the blade data is cached. The path is then rotated by using a timer and a rotate transform. Figure 7 shows the pump components for all the versions of the framework.

![Fig.7 WP7, BB and Android pump components](image)

2.5 The Valve Component

The framework implements 3 types of valves: 2-way valves, L 3-way valves and T 3-way valves. The valve control states differ according to the valve type. 2-way valves can be opened and closed, L 3-way valves can be left, right and closed and T 3-way valves can be left, right, both and closed. Additional UI related properties were also added to specify the state colors. Figure 8 presents the Windows Phone 7 implementation of the 2-way and 3-way T and L valves.

![Fig.8 Valve components](image)

For this implementation the classes hold only the color settings. The UI is completely independent, being specified in the control template. The colors, valve positions and orientations are displayed by using data binding and value converters and the visual state manager.

The Blackberry valve classes handle the UI drawing in code, in the control’s paint method. This limits the flexibility of the control in case the user wants another UI. In this case the user will need to derive from the valve classes and override the drawing code. In the Blackberry version the valve positions are calculated manually because the Blackberry framework does not support rotation transforms.

For the Android version the valve drawing is also done in code, but this time the valves use rotation transforms to rotate the corresponding parts in order to obtain the desired result.

3 Framework flexibility and customization

The components in all framework versions can be customized to a certain degree (by setting the various control properties) by using the current implementation. The following paragraphs describe additional flexibility considerations.

By far, the version that offers the biggest customization potential is the WP7 version of the frameworks. This is due to the “lookless controls” philosophy behind Silverlight [1,3,4]. Controls developed with the Silverlight framework do not tie the behavior to the UI. The UI can be specified declaratively and almost independently from the control behavior. This was seen in the previous sections when we discussed the various WP7 components. Besides using only control templates to change the appearance of the controls one can also use custom value converters.

Figure 9 presents a set of gauge controls that use custom value converters to achieve some radical design. The other WP7 components can be changed in similar ways.

![Fig.9 customized WP7 gauge controls](image)

The Blackberry and Android versions are a lot more limited. This is mainly due to the fact that the components’ appearance is specified in code and is tightly coupled to the control behavior [1,2]. If a developer wants to customize the components in a way that is not supported by existing properties he
will have to override the component code. In order to do this the developer will need to have an understanding of the control structure. This alternative is no good especially considering the fact that the developer might not own the code. Figure 10 presents some customized Blackberry components.

![Fig.10 customized BB controls](image)

In order to make these frameworks more flexible, the UI drawing logic would need to be moved from the gauge controls to the ticks. The customizations in figure 10 were possible because the tick drawing was taken out of the gauge and moved closer to the tick representation by using a dedicated ScaleTick class. Figure 11 presents the class structure.

![Fig. 11 The ScaleTick class](image)

4 The practical application

In order to test the usefulness of the framework we designed an application that makes use of the framework controls in order to monitor the heat distribution system installed in a university campus. This application is a mobile client that runs on all the mobile platforms the framework supports. The application gets the sensor data via a WCF web service interface. Using such a service interface allows us to abstract the different communication protocols from the client.

The university’s heat distribution system is split into several independent areas that cover the student dorms as well as the university buildings. A monitoring diagram was built for each of these areas.

Figure 12 presents the main application screen. This screen contains all the monitored regions and is used to navigate among the different areas.

When the user selects one of the areas he wishes to monitor, the user is taken to the corresponding diagram. This is a high level diagram that shows an overview of the system. These kinds of diagrams can be built using the framework’s pipe, tank, valve and pump controls. Figure 13 presents such a diagram.

From this high level diagram the user has the possibility to view additional details. For example, the user can click one of the numeric indicators in order to see more details about the data that is coming from a particular PLC as seen in figure 14. This screen has 3 main areas. The upper left corner shows a radial gauge that presents the current value for a particular PLC and its limits. The upper right corner presents the minimum, maximum and average values measured since the screen was shown. The bottom part of the screen presents a graphic with the last 20 values that were measured for a particular PLC.

The images presented in the previous 3 figures showed the Blackberry client. The Android and WP7 clients have the same features as the Blackberry version.

![Fig.12 The application main screen](image)

![Fig.13 High level diagram](image)
5 Conclusions

Developing SCADA clients can be a difficult task. This is especially true when it comes to developing clients that need to run on mobile devices. In the past these kinds of clients were either text based or had very few graphical components. This was mainly due to the poor hardware and framework support on those devices. Recent years have seen a huge expansion in the mobile device area. Among other things, this resulted in powerful frameworks being supported by the various devices available.

The framework presented in this article helps developers build SCADA user interfaces more easily by providing them with a set of common components. Developers don’t need to spend time building and testing their own components in order to use them in the SCADA HMI. Instead they can use this framework and shift their focus to the required business logic and to other problems.

The framework supports powerful and flexible gauge controls, easily customizable tanks, valves and pump controls as well as some pipe controls. Future versions will include chart controls and alarm management controls.

One last thing to mention is that even though this framework helps speed up the UI development, on some platforms this isn’t enough. One example is when building a Blackberry SCADA client. The main drawback with Blackberry is that the out of the box SDK doesn’t support a visual designer for building UIs. This leads to developers having to spend a lot of time tweaking the UI in order to arrange the components just right. The problem gets even worse if you consider the fact that you need to start the emulator every time you want to test the smallest of changes and it takes a lot of time for the emulator to start.

This problem could be mitigated by building a graphical designer that can be used to generate Blackberry code files. Such a tool might help Blackberry developers build the diagrams faster by allowing them to view the results while adding the components. This is a subject for future work.

References:

[1] Pro Android 2 - Sayed Hashimi, Satya Komatineni – Apress