A LIGHT-WEIGHT MVC (MODEL-VIEW-CONTROLLER) FRAMEWORK FOR SMART DEVICE APPLICATION

Budi Darma Laksana
Alumnus School of Information Technologies, The University of Sydney
email: blak4451@mail.usyd.edu.au

Cherry Ballangan
Faculty of Industrial Technology, Informatics Engineering Department, Petra Christian University
e-mail : cherry@petra.ac.id

ABSTRACT: In this paper, a light-weight MVC framework for smart device application is designed and implemented. The primary goal of the work is to provide a MVC framework for a commercial smart device product development. To this end, the developed framework presents integration between the classic design patterns, MVC and state-of-the-art technology XAML by adapting a MVC framework of an open source XAML efforts, MyXaml into .NET Compact Framework. As the compact framework only comprises 12% of .NET Framework library, some design and architectural changes of the existing framework need to be done to achieve the same abstraction level. The adapted framework enables to reduce the complexity of the smart device application development, reuse each component of the MVC separately in different project and provide a more manageable source code as the system architecture is more apparent from the source code itself as well as provide a commonality of the development pattern. A prototype of simple database interface application was built to show these benefits.

Keywords: Model-View-Controller (MVC), XAML, .NET Compact Framework

INTRODUCTION

MVC pattern is one of common architecture especially in the development of rich user interactions GUI application. Its main idea is to decouple the model which encapsulates the states of the application and the view which is GUI representation of a model. The interaction between the view and the model is managed by the controller.

In broad term, constructing an application using an MVC architecture involves defining 3 classes of modules:

- *Model*: The domain specific representation of the information in which the application operates
- *View*: User Interface element such as HTML in web applications or Windows in the desktop
- *Controller*: it is an event listener that responds to an event and changes the model or the view. [13]

With the proliferation of smart devices that have evolved from a text-based small pocket sized organizer to a rich graphical unit interface device that can be use for a lot of purposes e.g. games, email, messaging or even video-streaming, it is very beneficial to implement the pattern in the development of smart device's application. In the past, the development of smart device application required special skill of embedded operating system and also most of them written in C/C++ programming language. The .NET Compact Framework is Microsoft's approach to overcome this steep skill acquisition curve by integrating the framework into Visual Studio .NET and providing the developers with the same programming languages and environment development. However because of the limitation of the smart device such as processing capacity and memory, .NET Compact Framework [1] only comprises 12% of the .NET full framework, which means the compact framework only contains a subset of the full framework libraries.

The current popular implementation of MVC pattern development framework such as Spring MVC for Java [11] and Microsoft User Interface Application Block for .NET framework [12] are too heavy to run on a smart device. But recently, Microsoft is developing an XML based Markup language called XAML[2] (pronounced "Zammel") ,eXtensible Application Markup Language, the next version of Windows GUI codes name Avalon. The usage of XAML for windows is similar to that of HTML for web pages in which the user controls, images and layout are defined declaratively. This leads to separation of user

interface (UI) represented by XAML and the program codes that manage the user events; a similar analogy can be seen in the html pages with their associated java /asp scripts. Currently the available XAML products for .NET are UIML (User Interface Markup Language) [3], Microsoft XAML, Xamlon[4] and MyXaml[5], and only Xamlon Mobile and MyXaml are applicable for smart devices. Unfortunately Xamlon Mobile is a commercial product and is still in the beta release, on the other hand, MyXaml is open source, despite the fact that its main design is for .NET full framework, there were some efforts from a Dutch developer, Bertje Bier, to port MyXaml into .NET Compact Framework (MyXamlCF), his work were published in codeproject.com [6].The compact version has very limited capabilities but it still opens to further extensions.

MyXaml

XAML is a serialization format of object graph instances with the potential for making cross platform UI definition possible regardless of platform and language. It allows the programmer to separate the UI definition from underlying business logic and offers the possibility that a single UI definition can be used on different platforms. Moreover, XAML allows the users to edit the presentation layer without requiring the usual development tool or programming language.

MyXaml is an open source implementation of XAML and according to its creator, Marc Clifton, MyXaml is a general class instantiator capable of initializing properties, wiring up event to event handlers and drilling down into property collections [5]. These functionalities allow to instantiate the presentation layer components such as forms and controls through the mark up rather than code. Furthermore, it is able to instantiate any objects provided with namespace declaration that maps to the .NET assembly namespace in which their classes' definition reside.

The core interpreter of MyXaml is the XAML parser, represented by MyXaml.Parser class. At runtime a generic class instantiator in the MyXaml.Parser will parse the xml file, and using reflection it will instantiate any classes declared in the file, initialize values and object to the properties of those classes, add items to collections and wire up events to inline code and supplied event target objects. Moreover, there is a compliance rule for a class to be able to be instantiated by MyXaml, it must have parameter-less constructor and uses public property get/set method to interact with the different properties of the class.

MVC Model in MyXaml

MyXaml provides a comprehensive framework of MVC design pattern. The model is in MyXaml.MxContainer assembly and the controller is in MyXaml.MxEventPool assembly. Figure 1 shows the MyXaml MVC Model.

The model might act as a data binder similar to that in the windows data binding or it might be a value object that represents the state of the application. Below are the model main classes:

- MxProperties represents a generic the properties of the model. It has onValueChanged event which is fired each time its value changes.
- MxDataContainer is the main core of the model in a form of generic data containers that contains a generic set and gets method.
- MxDataBindingModel is class wrapper of windows data binding that might bind windows' control in the view and the model directly.
- MxContainer is a class that generates MxDataContainer subclass, instantiate and initialize it during run time.

The controller, MxEventPool class, have a clean controller that does not maintain any states of the view and it does not have any knowledge about the control in which the event is generated; consequently it is pure business logic and only manages the model state. This creates a better architectural pattern as the view states are not managed by the controller. Figure 1 shows the controller main classes:

- EventHelper is a generic and defensive event publishing in which it can fire any kinds of event delegate dynamically.
- MxEventPool manages many of different delegates and their associated events as well as event handlers. Events are abstracted by associating the event handler with a tag. And the events are invoked by referring to the tag

(rather than the event delegate). It also support the runtime generation of the codes when ControlEvent is used for mapping between a Control and a tag (refer to event).

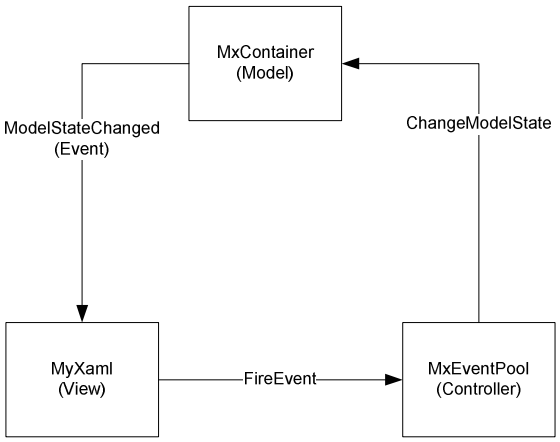

Figure 1 MVC Framework in MyXaml

- ControlEvent is a mapping between a reference of a Control and an event.
- Subscriber is a class that provides a unique name for each event handler. The view will call the event handler based on this name.

This framework provides a clear architectural distinction among the view, the controller and the model.

MyXaml compact version has subset functionalities of the full version owing to the limitation of .NET Compact Framework. These are the missing features that affect the adaptation of MyXaml into compact framework:

It is not to do any run time compilation though it is the common way to generate code in run time in MyXaml such as in the case of MxContainer and MxEventPool where both relied on this mechanism in a considerable degree.

- It does not support CreateDelegate and DynamicInvoke, these 2 methods are the main handler in MxEventPool and also in the Parser for wiring the event and its event handler. And also required by MxContainer to wire container events to the view listeners.
- It does not support TypeConverter for converting from one type to another.
- It does not support ISupportInitialize interface. Most classes in the MyXaml assembly implements this interface. As MyXaml will call BeginInit() and EndInit() methods in the objects initialization.

The "NET Compact Framework" [10] provides a complete reference of its functionalities and limitations.

Due to this limitation, the MyXaml MVC model needs to be adapted to be able to work properly in .NET Compact Framework.

ARCHITECTURE AND DESIGN

The adaptation process comprises: the assessment of existing architecture from the .NET Compact Framework point of view, redesigning the architecture to comply with the framework and finally rewriting the source codes based on the new design.

Controller

The *MxEventPool* consists of a dictionary collection of event delegates in which each delegate in it is identified by a special key (tag), and they can be combined to create a chain of event handlers. The events are fired through an *EventsHelper* class that dynamically call the delegate's *DynamicInvoke* Method. There are 2 important fields in the pool class, which are *SubcriberList* and *ControlEventList*.

SubscriberList is a collection of *Subscriber* class that has map the event name (*eventName*) and event handler method (*handlerName*) in the pool object *(instance*), or in other words each event handler in the pool might be identified by a unique name. Another functionality is the *ControlEvent* class to map a Control's events such as *Button's click* or *TextBox's TextChanged* to the event name in the pool. Figure 1 shows the class diagram of the controller framework.

To start the adaptation process, first of all it needs to compile the existing classes in the .NET compact framework environment. The compile error that occurs during compilation happens in:

- The *EventHelper* class, the compact framework does not support Dynamic Invoke method to call on the delegate dynamically
- The *EventPool* class, the automatic code generation part causes the compile error; and the creation of delegate for each subscriber also produces the error as the compact

framework does not support *CreateDelegate* methods.

The 'dynamic invoke' problem can be addressed by replacing the delegate dynamic invoke by calling the delegate directly as shown but the delegate creation of each subscriber by calling *CreateDelegate* methods might be addressed by declaring a delegate directly and creating a new class that represents its event handler. In this case I declared a delegate named *MulticastDelegate* and created *DelegateHelper* class, which is passed into constructor of the *MulticastDelegate* .

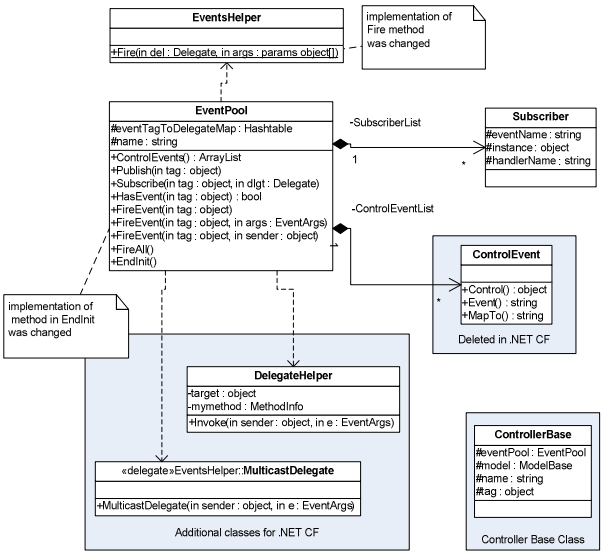

Figure 2 Static Diagram of compact Controller Framework

However, the automatic code generation requires for each defined *ControlEvent* object. And the purpose of this object is to map the Control's event property to any subscribed event handler in the pool declaratively. From my point of view this mechanism of declaratively mapping controls in a window form and their event handlers in the event pool is not a good design approach as the event generated by the controls might be one of two types: an event that changes the state of the model or an event that changes the state of the view. In the previous case the event pool can be a bridge between the view and the model, but in the latter case, it is a poor design to put the handler in the event pool. Provided there are two types of events generated by the view, the code will be more maintainable if both types of events are handled by the view itself imperatively and the view will dispatch the event of the first type to the event pool. Based on this rationale, I did not assimilate the *ControlEvent* class and its automatic code generation into compact framework.

In addition the MyXaml Event Pool, a controller base class is added into the framework Figure 2. All controllers should be derived from the base class. To use the framework, a *MVC.Controller* class that derived from *ControllerBase* and a Model object named *TheModel* are provided. The event pool has one event handler: *PersonHandler* and there are 3 event names (*AddPerson, DeletePerson* and *AddAddress*) that invokes the handler. The View base will use the event name to invoke the handler in the event pool.

Model

The model in MyXaml MVC implementation relies heavily on runtime code generation and compilation, in which data model is represented as a subclass of *MxDataContainer* that generated on runtime by *MxContainer* class.

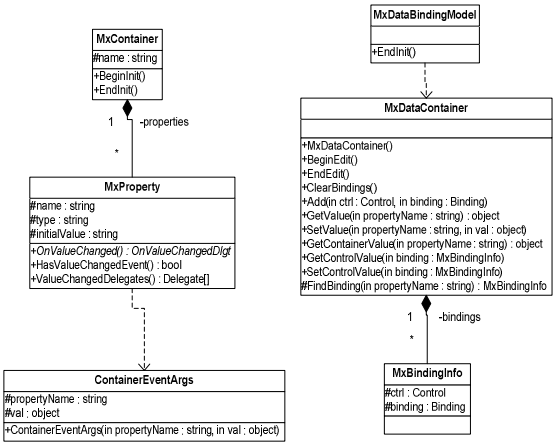

Figure 3 the MyXaml Model framework

Unfortunately, the .NET Compact Framework does not support run time code generation, so it requires different approach to have the same abstraction level as that of the full version. Consequently, instead of implementing the code generation, I use a *Hashtable* to represent the object property-value. In this approach, the key of the *Hashtable* corresponds to the property name and its value is associated a *PropertyHelper*

instance. It is a custom class that represents an object and its delegate. As the each declared property also generates an event, as an example the Name property has its *OnNameChanged* event. In *Hashtable* version, each key in the collection would correspond to a *PropertyHelper* object, an object that constitutes a property-value and an event delegate.

Despite of the fact that there is no run time generation of an object that derived from *MxDataContainer*, the *MxContainer* class is not required. So in the compact version implementation, there is not *MxContainer* class and the *MxDataContainer* can be declared directly in the XML. The implementation of *MxDataContainer* also needs to be changed, in the full version *System.Reflection* facilitates all invocations to its derived object's property but in the compact version the approach is simpler as all the properties values are resided in the *Hashtable*.

The issue with the Hashtable version is how to deal with the event delegates such as how to create the delegates and fire it when there are some events. An event in the full version is fired when a property's value in the container changes as well as when the control's property bound to the container's property changes. The full version uses the window's data binding to bind the control and the container's property and use *System.Reflection* to changed the property values when some events occurs. The windows data binding is not fully supported in the compact version. So data binding architecture in the compact framework is not a good approach, that instead of data binding, I use event to convey messages of property changes between control and its data bound. This approach produces a new complexity and requires complete changes of the *MxDataContainer* implementation as well as need some additional classes. The model of Compact Framework is depicted in figure 4.

DataContainer is the central class of the architecture, it has direct association to the *MxProperty* class; and the delegate class, *OnValueChangedDelegate*, is declared in the namespace scope, instead of in the class scope. In addition, there are more methods in the *DataContainer* than those of the *MxDataContainer*. The same case also happens in the *MxProperty* class.

The *MxBindingModel* is replaced with *Binder* class, as the previous is the sub-classed of *BindingModel* that implements the windows data binding, the latter instead of using windows' data binding it uses the new method, *SetEventHandler* which is called from a helper class called *Binder*, in the *DataContainer* class. The *ContainerEventDelegate* is the delegate class that handle the *OnValueChanged* events; this must be declared as a separate class as MyXaml CF does not support other event handlers than *EventHandler* Delegate. In summary, all the new methods and classes are used to deal with the event passing between controls and *PropertyHelper* object in the *DataContainer*.

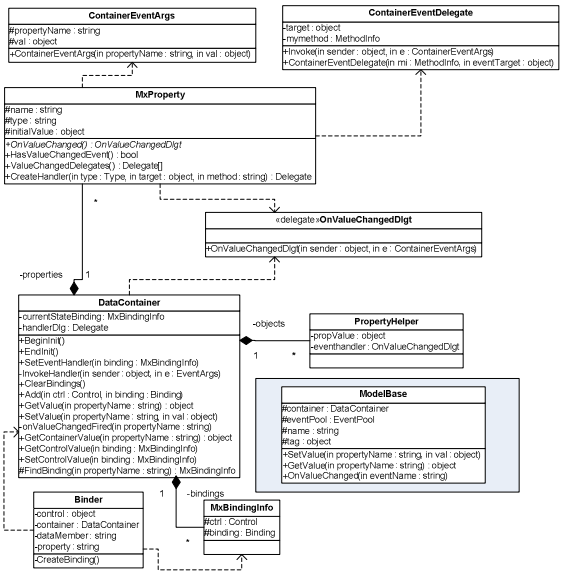

Figure 4 the model framework for compact version

ModelBase is the base class of the model, the class encapsulates *DataContainer* and *EventPool*, and so the model might interact with the event pool through *OnValueChanged* method which is an event handler for *OnValueChanged* events. It handle the *ContainerEventArgs* and pass a generic event to the event pool.

View

The view consists of 2 parts: the declarative part, which is the XML file declaring all the controls in the forms and the event handler of each controls that declared imperatively. The declarative part is a normal MyXaml file but the imperative one must

inherit from a *ViewBase* class. The class provides encapsulation of an *EventPool* and does a simple interaction with the working parser. It also contains a form property, it associates to form instance in the object graph in the parser.

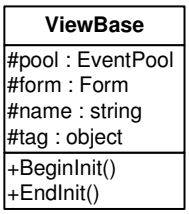

Figure 5 View Base class

Deployment View

The structure of the application deployment consists of declarative part (XML) and imperative part (C#). The declarative part is for initializing the GUI controls and the framework classes; where the declarative part is for the business logic and the interaction among the framework classes. Figure 6 is the deployment structure of the compact MVC model.

The Main XML only has 'include' statements; it refers to View XML that declares all GUI controls, Controller XML that declares the event pool and Model XML that declares *DataContainer* and the *Binder* classes. The imperative part composes of a main application class that load the main XML (Fig. 6). In addition View class, Controller class, and Model class should inherit their associated base classes. These base classes implement the business logic based on event handling, in which the user interaction would be captured in the view class, and the event that changes the state of the model class will be passed to controller through the controller's event pool. And it will process the message and pass the result to the model dealing with the data state.

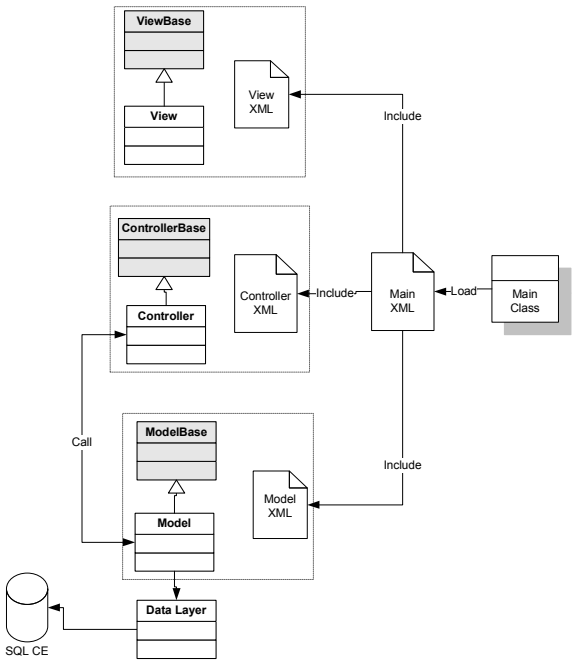

Figure 6 Deployment structure of .NET CF MVC model

The deployment structure is different from that of MyXaml MVC model deployment because in the existing framework it does not utilize the *Include* model. For more detail implementation will be discuss in the following chapter of using the framework to develop a simple application.

EVALUATION AND TESTING

Application Requirement

The application is a simple interface to customer table in the member database. User might add, delete, and search a record in the table. User Interface and database schema are provided in the following section.

Table 1 Member Database Schema

Database : Member		
Table: Customer		
Field	**Data Type**	**Remarks**
CustomerId	Int	Primary Key, Auto-increment
FirstName	NVarChar(20)	Not null
MidName	NVarchar(20)	Allow Null
LastName	NVarchar(20)	Not Null

Database schema

Table 1 shows the member database schema. It has a table named customer.

User Interface

The form enables user to interact with the customer table. It consists of textboxes as user entries, list view as the output as well as menu control for basic user interaction e.g. for adding, deleting and searching records But additional functionality might be added in the user interface such as simple validation, and fields clearing. Figure 8 shows the user interface design.

```
<?xml version="1.0" encoding="utf-8" ?>
 <MyXaml>
   <Form def:Name=frmMvc" >
     <Include Src=" View.xml"
ComponentName="view" />
     <Include Src=" Model.xml"
ComponentName="MemberModel"/>
     <Include Src=" Controller.xml"
ComponentName="MemberController"/>
   </Form>
</MyXaml>
```

```
public class MainClass
{
   public MainClass()
   {

              Parser parser = new Parser();

parser.LoadObject("Main.xml","frmMvc",null,null);
              ViewBase v = (ViewBase)
parser.GetReference("MemberView");
              Application.Run(v.Form);
            }

            public static void Main()
            {
              MainClass main = new MainClass();
            }
}
```

Figure 7 Main XML and main application class

Development cycle

The development cycle comprises 2 stages, Declarative development (to build the XML documents that capture the design requirement) and Imperative development (to build the C# code that handle the event passed among each system component).

Declarative Development

As mentioned in the previous section the XML based declaration consist of 4 parts:

- View contains all controls declaration, and their associated event handlers. It will be parsed and rendered by the MyXamlCF engine and as a result is a form display complies with the user interface design. In this case, the view will contain the declaration for 3 Labels, 3 *TextBoxes*, 1 *ListView* and 4 *MenuItems*. All the control properties are set in the XML. In addition to the control declaration, it also contains the declaration of the event handler class; in this application is the View class. So each declared event will be wired to each method in the class.

Figure 8 User Interface Design

- Controller contains declaration of the Controller object and its *EventPool* object with

its subscribers. The subscriber will match the defined Tag to a method name in the controller object.

- Model contains declaration of the Model object and its container. The container object represents the data state. In this case it constitutes of *MxProperties* objects match with the database schema and also additional data states required by the View such as a *DataTable* object as the ListView's data source, Row index for highlighting the selected row, and member id for changing the status of delete menu item. The additional properties have *onValueChanged* event that will fire when there are some property changes. The event will be captured by the *EventPool* and forward to the respective listeners in this case is the view. Moreover, there are also declarations for Binder objects. The Binder will bind each *TextBox*'s Text property with the data container. The binder is a custom version of the windows data binding.
- The last XML file is the main XML as it is the glue of all the other XML's. In the Main XML it is a necessity to have all namespaces declared in the other documents and also the form must be declared in the document. Inside the Form element are only Include elements to the view, the controller and the model XML documents.

The Imperative development

As the declarative parts only instantiates the object and assigns its properties, the imperative code requires to handle the behaviour of the application. It consists of 5 components:

- The Main code has main methods to instantiate the parser and load the main XML into it.
- The View class inherits the *ViewBase* class and contains the entire event handlers declared in the view XML. The event handlers only deal with control rendering and capturing user's gestures. It will pass the event that might change model state to the controller through *EventPool* object.
- The controller class must inherit the *ControllerBase* class. It contains the business logic of the application as it will call the model methods for each received messages from the view.
- The Model class should inherit the *ModelBase* class. It interacts with the data layer directly to change state of its data container. It has also atomic methods that might be composed into some business logics by the controller. In this application it has Add, Delete, Search and Clear methods.
- The Data Layer objects is a component that communicates directly with the database. It opens connection to the database and doing the query or update commands. In this application, the data layer is represented by *DBHelper* class.

Evaluation Criteria

The framework would be evaluated qualitatively based on these 2 criteria:

- The development point of view explains how the software is packaged and how it can be developed as well as contrasting it with the traditional development process.
- The architectural point of view explains how the framework adheres to the traditional MVC pattern particularly and design principle in general.

The Development View

From development point of view, a framework should ease the development process, simplify the software management and enable reuse and commonality. Therefore, the framework will be assessed based on these properties:

- Reduce complexity. This is achieved by separating the domain representation and its behaviour. The domain model is declared in the XML documents and their behaviour is coded imperatively. This division simplifies the development process especially in a complex system as the design properties are very obvious in the XML declaration. For examples: the view XML only contains the control declarations and the view C# codes contain the controls' behaviour in regard of the user's events. In contrast with the traditional approach when the domain model and its behaviour are coded imperatively. It was hard

to have an architectural view from the codes itself.

- Reusability. The loose-coupling between the domain model and its behaviour enable to reuse both components separately. In the toy application, it is possible to reuse the GUI (the View XML) in other application with different behaviour. Or reuse the model declaration for different database schema. The reusability of traditional approach is very limited as the GUI or the model is tightly couple with the event handler or the database schema.
- Manageability. Although in this development framework there are more managed entities (XML and C#) than those in the traditional approach, it does not imply that the new approach is less manageable. In contrary, the separation of XML declaration and its C# behaviour brings more clarity of the application architecture in the codes level resulting in better codes readability. In a complex system this advantage will undermine the inconvenience of maintaining more application entities.
- Commonality. It will accelerate the development process and have more assurance of quality design as developers follow the same proven development patterns and styles. In this thesis, the toy application complies to the MVC pattern providing by the framework from the beginning of the development process

Architectural point of view

In this section, the contrast between MVC architectural point of view of the traditional .NET approach and MyXaml approach is presented.

In .NET, the wiring up of the event requires that the code has both knowledge of the Control class (the view) and the instance (the controller) that handles the event. Any changes in the controller will also change the view and vice versa. In this framework approach, the Control is wired to an event pool that dispatching the user event to any subscriber that is declared in the Controller XML. So the wiring between the event's source and its sink is centralized in one XML file and it can be changed or rewire without affecting the View.

The controller represents a set of methods that manages the model states. In .NET implementation, as the event handler is tightly coupled with the view, it is impossible to implement MVC pattern, results in code rewrites when the View changes, impede the flexibility of the application.

In .NET there is no difference between the event handlers that change the view's states and those that change the model states because the view call the controller method directly. But in this framework the view consume any control events and provide the controller with a common type of event as it does not have any reference directly to the controller. In other words, the view and controller can be totally separated.

In .NET, it will result in a complex architecture when it requires separating the model that has some events wired back to the view. But in this framework the event wiring between model-controller is the same as that of view-controller. So there is only one point of change for wiring up events and their listeners no matter of their sinks and sources.

VALIDATION

The current version of the overall project, including developments listed above, has been extensively tested and validated, and as a result scheduled for commercial release during the first quarter of 2006.

CONCLUSION

The primary goal of this paper is to implement a lightweight MVC framework with the .Net Compact Framework for a smart device product. Consequently, the MyXaml MVC framework had been adapted to be fully functional in a limited resource environment.

This concluding chapter presents the contributions of the developed framework and followed by a discussion of future work.

Contribution

The paper makes the following contributions:

- First and foremost, it presents a MVC application development framework for .NET Compact Framework.
- It integrates the new development software paradigm based on state of the art technology, XAML, into .NET Compact Framework.

- It demonstrates the developed framework by building a toy application which incorporates use interface, business logic, and data representation as well as data access layer.
- It shows some benefits of the framework in term of reducing application complexity, code manageability, component reusability and development commonality.

Future Work

Although the framework provides an adequate separation between Model-View-Controller, it can be enhanced and furthered in various ways:

- Workflows framework. One of the advantages of the declarative programming is a workflow or script representing logic of an application. It simplifies the development process of complex business logic. Thus it is obvious that porting this framework to .NET Compact Framework would decrease the lines of codes requires in the Model and Data Layer.
- Adaptation of state machine framework that able to create an event based state machine declaratively. An advantage of this framework is the ability to manage application's states out of the code, in other words, separating the application states from the logic that determines the state transition. This would provide a more manageable application.
- Optimization. As the .NET Compact Framework has limited resources in terms of memory and processor speed, benchmarking is beneficial and performance improvement might be achieved by investigating the parser implementation.

REFERENCES

[1]. Smart device project: .NET Compact Framework. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_evtuv/html/etconNETCompactFramework.asp

[2]. eXtensible Application Mark-up Language (XAML) resources. http://msdn.microsoft.com/msdnmag/issues/04/01/Avalon/default.aspx

[3]. User Interface Mark-up Language (UIML) resources. http://www.uiml.org/specs/

[4]. Xamlon web site. http:// Xamlon.com

[5]. Project MyXaml: open source XAML web site. http://myxaml.com

[6]. MyXaml Compact Framework implementation. http://www.codeproject.com/netcf/UsingMyXamlCF.asp http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_evtuv/html/etcondifferenceswithnetframework.asp

[8]. Imperative programming style. http://en.wikipedia.org/wiki/Imperative_programming

[9]. XAML definitive guidance. http://blogs.msdn.com/seangrimaldi/archive/2004/06/12/154446.aspx

[10]. Wheelwright A., Wigley S, ".NET Compact Framework" . Microsoft Press, RedMond, Washington. 2003

[11] Java Spring MVC http://www.chariotsolutions.com/slides/Intro_to_Spring_MVC.pdf;jsessionid=1F4239D0E68D5ED91EC74B7443E8BF18

[12] Microsoft User Inteface Application Block http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/uip.asp

[13] Model-View-Controler from Wikipedia http://en.wikipedia.org/wiki/Model_view_controller