.NET As A Platform For Wireless Applications

Juha Järvensivu and Tommi Mikkonen

Institute of Software Systems, Tampere University Of Technology, Korkeakoulunkatu 1, 33720 Tampere, Finland

Abstract: Wireless applications implemented in mobile gadgets are a new trend in software development. One platform on top of which such applications can be implemented is Windows, where two different flavours of design environments are available. .NET Framework (.NET) is aimed at full-fledged computing environments, and it is used in e.g. laptops. In contrast, .NET Compact Framework (.NETCF) is for smartphones and PDAs that consist of more restricted hardware. From the development perspective .NET and .NETCF are closely related as they rely on the same application model. Moreover, .NETCF is a subset of .NET environment, with features that are not relevant in smartphones or PDAs removed. Therefore, it seems tempting to run the same applications in all wireless Windows environments, disregarding the type of the device. In this paper, we analyze the possibilities achieving this goal in practise.

1 Introduction

Wireless applications are gaining more and more interest due to the increasing number of devices. One family of such devices is based on Windows, where two slightly different platforms are available: .NET Framework (.NET for short) and its simplified version .NET Compact Framework (.NETCF). Of these platforms, .NET is aiming at devices that run on e.g. laptops. In contrast, .NETCF is targeted for smartphones and PDAs that run on more restricted hardware.

Despite being targeted to different types of devices, .NET and .NETCF are closely related, however. In particular, they share the same basic infrastructure and application model. Moreover, .NETCF is a subset of .NET, with only features that are not suited for a restricted environment removed. Therefore, it is tempting and in fact seemingly straightforward to develop applications that run unaltered in all wireless Windows environments.

This paper studies the possibility to design and implement such wireless applications. The study is structured as follows. Section 2 discusses the main commonalities and differences of .NET and .NETCF. Section 3 introduces some principal problems in designing wireless applications for .NET and .NETCF environment. Section 4 introduces a simple framework that allows common application development for all wireless Windows clients. Then, Section 5 introduces a sample application that has been implemented using the framework. Section 6 presents an evaluation of experiences gained when designing the application. Finally, Section 7 concludes the paper.

Järvensivu J. and Mikkonen T. (2005).
.NET As A Platform For Wireless Applications.
In *Proceedings of the 4th International Workshop on Wireless Information Systems*, pages 104-113
DOI: 10.5220/0002557801040113
Copyright © SciTePress

2 Overview of .NET and .NETCF

In this section, we give an overview of .NET and its smartphone and PDA capable version, .NETCF. The focus of the presentation is on the properties that are important for wireless setting.

2.1 Platform Principles

Perhaps the most important technical aspect of .NET platform is the definition of an intermediate representation for executables. This representation is called MSIL (Microsoft Intermediate Language). All composed .NET programs are first compiled to MSIL, and only immediately before execution, they are compiled to executables using Just-In-Time (JIT) compilation techniques. For a developer, this decision means that several programming languages can be used, assuming that a compiler is provided for generating MSIL from the used language. Furthermore, porting of the whole .NET system is eased, as it is enough to produce MSIL-to-platform compiler per hardware platform, not one for every supported language.

The use of MSIL and JIT requires support at run-time. This support has been implemented in software component called CLR (Common Language Runtime), which forms the runtime environment of .NET. The use of JIT compilation is optimized so that code is compiled the first time it is needed. After the compilation, it is stored in the memory in the compiled form as long as the program is active. As a consequence, the first call of a method is slower than subsequent calls. In addition to JIT compilation, CLR also handles memory management. It provides facilities similar to those of Java to prevent memory garbaging.

2.2 Web Services in .NET

One of the main elements of .NET is eased development of applications that use Web Services. By definition, Web Services provide means for accessing services offered by the Internet. In the sense of implementation, Web Services use the following techniques.

Web Services are accessed using SOAP protocol (Simple Object Access Protocol). A SOAP message is a text based XML document (External Markup Language) for exchanging messages between two pieces of software that may run in different computers. A SOAP does not define any protocol for transferring messages over internet but HTTP protocol (Hyper Text Transfer Protocol) is commonly used. SOAP message is a simple one-way message where the sender does not receive a response. The receiver could, however, send a response back to the sender. Then both pieces of software have to work as sender and receiver.

In the technical sense a SOAP message is an XML document, which contains a header element and a body element. The header element contains application specific information about SOAP message and the body element contains the actual SOAP message.

WSDL (Web Service Definition Language) is an XML document that defines Web Service's interface offered to the client. WSDL contain all the information required for writing a client for a particular Web Service. It is specifies the location of the service and the operations the service exposes.

2.3 .NET Libraries

Libraries of .NET can be divided to four categories. The first category includes all basic classes needed in every application. This category is called BFC (*Basic Framework Classes*). Such classes include *Object, String, Thread*, and *Type. Object* class is the root class of the class hierarchy of .NET. It provides some low-level services to derived classes. Similarly, *Type* is the abstract root class of the reflection functionality and is the primary way to access metadata. *Type* is used to get information about the type declaration, such as constructors, methods, and events of classes.
The second category is called *ADO.NET*. *ADO.NET* is a set of classes that expose data access services. *ADO.NET* is an essential part of .NET providing access to relational data, XML documents and application data.
The third category is discussed about *ASP.NET*. *ASP.NET* allows the design of web applications and provides support for Web Services described above.
The fourth category is called *Windows.Forms* that contains classes for implementing graphical user interfaces.

2.4 .NET Compact Framework

.NET has been squeezed to smaller size under the name .NET Compact Framework (.NETCF). This has required the removal of several libraries and other facilities that are not essential in smartphones or PDAs.

Based on the target devices and their requirements, ASP.NET has not been included in .NETCF, as it has not been considered adequate for restricted devices. However, other categories of libraries are included, at least to some extent. While the libraries are in principle included in the platform, there are some challenges to be tacked that need explicit attention in practice.

3 Principal Problems

In the following, we discuss some principal problems that prevent the use of same software in wireless devices disregarding whether they rely on .NET and .NETCF in their implementation.

3.1 Graphical User Interface

Perhaps the most obvious challenge regarding the compatibility of a wireless application results from user interface. Some devices contain a full qwerty-keyboard, some others are to be used with a pen, and some are to be used with one hand only. As a result, graphical components and especially their convenient layout vary, together with the pattern of user interactions needed for using the device. In the case of .NET and .NETCF, the following differences have been identified.

In both .NET and .NETCF environments, the user interface layout is typically designed with the Dialog editor integrated in Visual Studio. Unfortunately resource file format of .NET and .NETCF are incompatible. The difference lies in allowed serialized types. When a type is a serializable, it means that its state can be saved and restored. .NETCF only allows a fixed number of types to be binary serialized, whereas .NET is fully extensible.

Regarding the actual application design, the main difference in GUI programming is that .NETCF does not support all user interface components provided in full .NET. It is also important to note that if the same class exist in both .NET and .NETCF, this does not mean that .NETCF version includes all the methods, properties and events from the full version. Sometimes a class is not implemented at all, and sometimes it is only partially implemented. When composing a program, the designer that is familiar with .NET but not with .NETCF is bound to hit such classes frequently. As an example, Table 1 lists public methods of ListView class in .NET .NETCF.

Table 1. ListView public methods not supported by .NETCF

ListView	**.NET**	**.NETCF**
ArrangeIcons	YES	NO
BeginUpdate	YES	YES
Clear	YES	YES
EndUpdate	YES	YES
EnsureVisible	YES	YES
GetItemAt	YES	NO
GetItemRect	YES	NO
Sort	YES	NO
ToString	YES	YES

To add complexity to the design of a user interface, smartphone does not support all components provided in .NETCF. One such component is *Button*. If an instance of class *Button* is used in the code executed in a smartphone, it will compile successfully. However, as soon as CLR tries to enter the illegal code, a *NotSupportedException* will be thrown. Other similar classes supported on .NETCF but not on smartphone are e.g. *ListBox*, *TabControl*, *Toolbar*, *Statusbar*, and *ContextMenu*.

Further restrictions result from the fact that Smartphone UI is optimized for ease one-hand operations. Therefore it does not support a touch screen or any similar mouse activity. The maximum number of top-level menu items is two and only second top-level menu item can contain submenus. While .NETCF as such does not enforce these restrictions, exception *NotSupportedException* will be thrown at runtime if code does not follow them in practice.

3.2 Data Communications

In addition to the graphical user interface, different types of devices have different data communication capabilities. For instance, a modern GSM phone enables the use of GPRS data communication, whose data rate is 10-20kb/s. In contrast, a PDA may have a built-in wireless LAN card, whose capacity is e.g. 11Mb/s. Furthermore, possibilities of applying data communications offered by the platforms may also vary.

In our experiment, we have identified the following aspects regarding communications. Firstly, lower data rates slow execution. Secondly, serialization, i.e., the possibility to transfer complete objects to data and then back, is not offered in .NETCF. This is surprising, as this facility is needed for the use of Web Services, which are supported also by .NETCF. However, assuming that we rely on higher-level services and operations on the web, these problems can be circumvented.

We found no problems when using the same source code to access Web Services in different kinds of environments. Clients access Web Services in the same way, and automatically generated proxy-classes are identical.

4 Unifying Framework

In order to enable the use of the same source code in all wireless environments, we have implemented the following wrapper components. In addition, there are some issues that differ from normal Windows practices, which will also be addressed.

4.1 Graphical User Interface Design

The core problems of the graphical user interface design is how components are arranged to the dialogs, and how top-level menus and button components are handled on smartphone. It is also important to follow user interface programming guidelines and styleguides to ensure the high level of usability.

Unlike laptop version, user interface components can be automatically arranged in smartphone and PDA versions in a straightforward fashion. To accomplish this, a new class, *WrapperForm*, has to be created. *WrapperForm* is derivered from class Form defined in System namespace. As a result, user interface design has to be carried out only on laptop version, and after that, the wrapper layer handles layout automatically for smartphone and PDA versions.

Similarly, when creating a *Button* inside the wrapper layer, it is possible to replace buttons with menu items on smartphone version. To maintain compatibility with other platforms, the buttons have to be placed at the bottom of dialogs on other platforms too. This differs from normal windows behaviour at code level only in the way how Button component is created.

Restrictions of top-level menus of the smartphone can also be handled with a wrapper layer. This problem can be solved so that in case of smartphone an extra menu item, *Menu*, is created and all top-level menus are added to its submenu. This

only alters the way top-level menus are created. In particular, it is still possible to freely create any kind of menu structures.
The solution for these problems is to introduce a wrapper layer that handles the differences of the platforms. The resulting design is illustrated in Fig. 1.

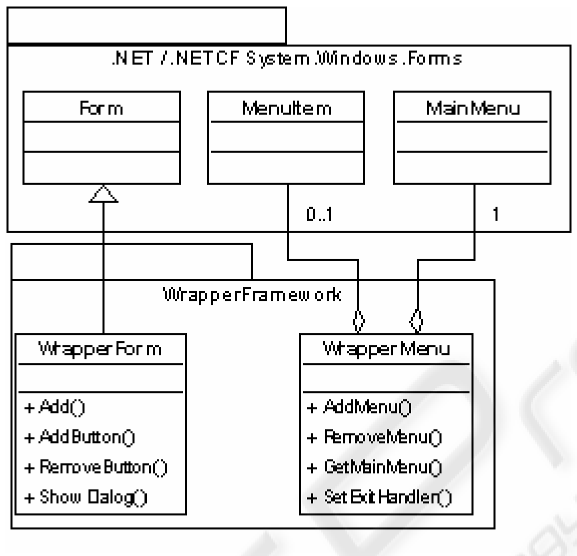


Fig. 1. Unifying wrapper layer

4.2 Internal Functions

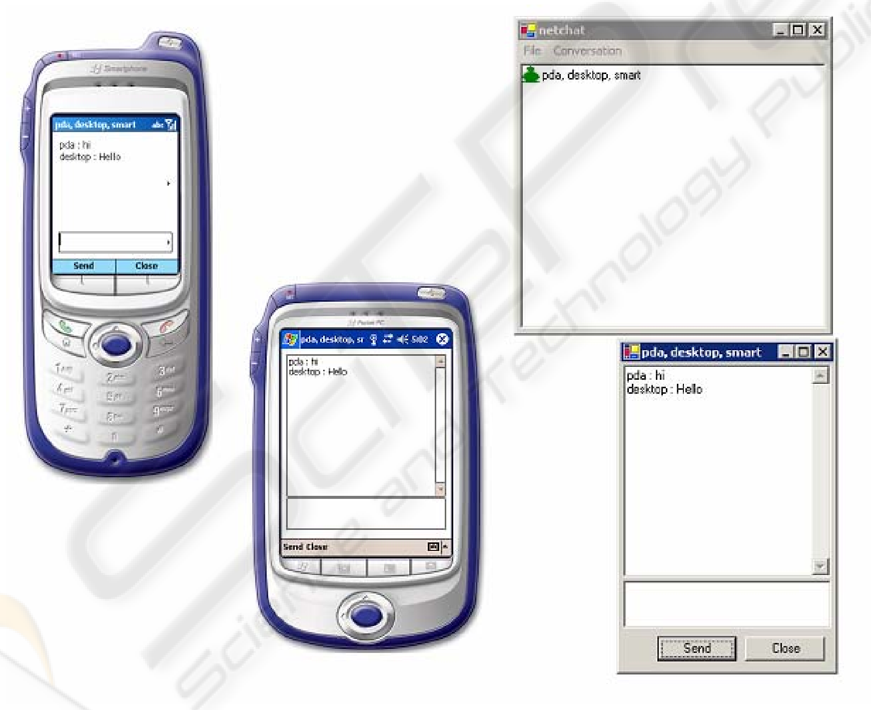
In the design of internal functions of an application, it became evident that the omission of methods and parameters for decreasing memory footprint also leads to compatibility problems. Many of the problems were identified when composing the actual code. As a fix that can be created when implementing a framework, we could implement a wrapper layer similar to that of the graphical user interface. However, such intensive wrapping seems superfluous and is dangerous for memory usage point of view. Therefore, we chose not to implement such a facility.

As all-purpose wrapping was not implemented, we had to search for other means of portability. As .NET environment favors the use of Web Services, we considered that this technology should be used as the means of portability. Thus, the structure of applications resembles that of Model-View-Controller pattern, where the model is not at the same computing unit as the rest of the system but distributed to a web service in the fixed network.

5 Sample Application

As a proof of concept, we have implemented a simple wireless chat application called NetChat that runs unaltered on laptops, PDAs and smartphones (Fig. 2). NetChat is a chat-program where users can create conversations and send text messages to each other similar to Microsoft MSN Messenger. The architecture of the application is illustrated in Fig. 3. The purpose is to implement one source code that can be used in different types of client devices. As already discussed, web services play an important part of the solution.

NetChat application consists of a Web service that forms the server-side and applications that run on mobile clients. The applications communicate with each other using XML Web service. Essential parts of the client-side application are graphical user interface code and XML Web service proxy-class automatically generated by wizard. In addition to the Web service, a basic file handling is needed to read and write some settings to the local text file.



pda : hi
desktop : Hello

Fig. 2. NetChatGUIs

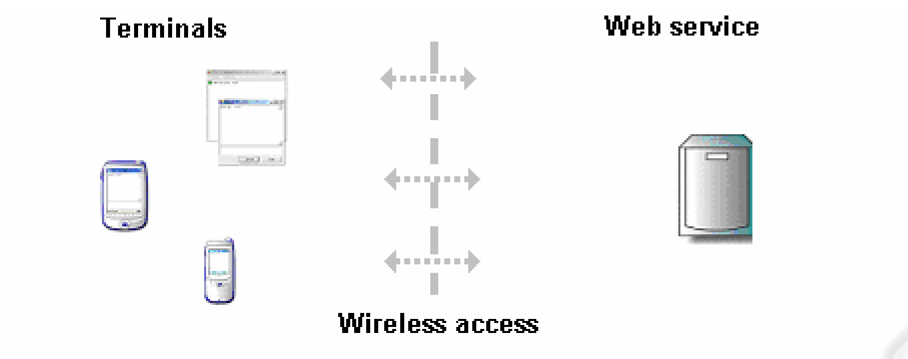


Fig. 3. Architecture of NetChat

5.1 Wireless Client

Using Web services is straightforward. The same proxy code can be used in different types of client devices without any changes. Only the required dynamically linked libaries have to be included in the project. It does not matter whether synchoronous or asynchronous methods are used. However, due to different data communication capabilities, some methods have to be implemented as asyncronous methods to avoid freezing of the UI on smartphone and PDA devices.

Many kinds of UI components are needed to implement the UI of NetChat application. The most important components are *ListBox* and *TextBox* components. The layout of the components is not any problem thanks to automatic layout functionality implemented on wrapper layer described in section 4. Also the use of simple components, like *TextBox*, is similar, even if they look very different on different types of devices.

In contrast, the circumvention of *ListBox* component is much more complex. Firstly, *ListBox* component is not implemented on smartphones in the first place, and therefore a *ListView* component has to be used. Selecting a single item caused problems, because multiselect property is not supported by .NETCF, but is by default set to true on .NET version. The application cannot change the value of multiselect property on .NET, because it is not implemented on .NETCF. Multiselect property has been removed from .NETCF because it is not needed on smartphones or PDAs. To solve this problem the number of selected items has to be counted and do the action only if exactly one item is selected.

The second problem was the background colour of a *ListView* item. In .NET version, it is possible to change the colour of individual *ListView* items. With the aid of this functionality, it would be easy to inform user about new messages. Unfortunately only changing the colour of complete components is supported on .NETCF.

However, both versions support the use of icons to identify the type of listview item. A purple icon was used to inform that a new message has arrived to the conversation. The same size of icons can be used on .NET and .NETCF versions. The next thing to do is to determine the path of application directory so that icons can be loaded. Default path cannot be used because it is pointing to different locations. In .NET version, it is pointing to the same directory where the program is started. In .NETCF version, it is pointing to the root directory of file system. Fortunately, it is very simple to get path to the executed program file and put icons to that directory.

5.2 Fixed Network Server

The main purpose of the fixed server is to store the data of NetChat application and to dispatch messages between users. Data can be stored as session objects or application objects. An application object provides a mechanism for storing data that is accessible to all clients, whereas a session object cannot share data between clients. Therefore the data of NetChat application have to be stored as an application objects.

Due to the implementation of Web services technology, it is not possible to send notifications from the server to a client when a new message has arrived. One possible solution is that clients poll the server and ask in a loop until new messages arrive. This is a very ineffective implementation and should be avoided if possible. The primary problem is how to inform the client when new message is arrived. The answer is the use of asynchronous web service calls. In the client side application, asynchronous web service method is implemented with threads and callback functions. It is a powerful way to create responsive user interfaces and function calls do not have to respond immediately. The user can ask a new message with a function that does not return until new message has arrived.

Because the fixed server is used over the Internet, it is not certain that the user always logs herself out when closing the NetChat application. Therefore, a mechanism to recognize invalid users is needed. In a NetChat application, the user has to make at least one function call to Web service every minute to keep connection alive. If the frequency of calls is less than one call in every minute, the server will destroy the user information. As the result, the user has to login again.

6 Evaluation

There are several reasons for problems to run same code in both .NET and .NETCF environments. The most obvious reason is the different patterns of usage. Smartphones are designed to be used with one hand only and they do not support touch screen or any other similar mouse activity. On the other, hand PDAs support touch screens but do not contain proper keyboard whereas laptops contain full qwerty keyboard and versatile pointing devices. A further major challenge is created by the differences in screen resolution.

From the technical point of view, these differences are not very serious, because the same messages are sent in spite of how components are used. Nevertheless these differences have to be taken care of in the design of a user interface. Because of the lack of pointing device, smartphone also does not support drag-drop functionality or *Button* component. Also the number of top level menus is restricted into two.

The second significant problem is that .NETCF does not support all user interface components provided in full .NET. Even if the same class exist in both .NET and

.NETCF, it does not mean that .NETCF version will have all the methods, properties and events from the full version. Methods and classes has removed for decreasing the memory usage. These differences are typically noticed only when implementing the application and thus can cause harmful problems. Fact is that many classes and properties have been removed if they are not needed in mobile device without thinking an influence at the compatibility. This kind of incompatibility is not only the problem of GUI libraries but can be found in other libraries as well. A pleasant exception is Web service. In general ADO.NET, including web service and database support, is well supported by .NETCF, apart from the object serialization.

7 Conclusion

This paper evaluates the option to use the same applications in both .NET and .NET Compact Framework environments. The main conclusion is that in the present shape, these platforms do not lend themselves easily for the design of common applications. There are several reasons for problems. The most important ones are:

- different patterns of usage
- different facilities for designing user interfaces (e.g. some components missing or unusable, incompatible resource file format)
- incompatible interfaces, resulting from the removal of some methods and their parameters in many .NETCF classes.

While the last item can be fixed in future releases of platforms, the other two issues are fundamentally associated with the different purposes of wireless devices.

Even if it is technically possible to circumvent all these issues as described in this paper, the resulting applications are not necessarily well-suited for end users, and therefore should be depreciated. However, for e.g. experiments on using a certain technology in a new environment or context, the commonalities offer a practical way to compose rapid implementations.

Based on our experiences described above, the most tempting wireless application development approach for us is to implement an application and device specific graphical user interface for all different types of devices if necessary, but rely on the use of web services for main functions of the application whenever possible. This approach is supported by both .NET and .NETCF development tools, which enable an easy access to web services from source code.

References

1. Ballinger, K., .NET Web Services: Architecture and Implementation with .NET, Addison-Wesley, 2003.
2. Järvensivu, J., *.NET as a mobile application platform*. Master of Science Thesis, Department of Information Technology, Tampere University of Technology, 2005. In Finnish.
3. Troelsen, A., *C# and .NET Platform*, Apress, 2003.
4. Wigley, A, Wheelwright, S, Burbidge, R, MacLeod, R and Sutton M, *.NET Compact Framework*, Microsoft Press, 2003.